MODUL PRAKTIKUM
MACHINE LEARNING & AI

DISUSUN OLEH:
Adi Muhamad Muhsidi, S.SI., M.Kom.

PROGRAM STUDI BISNIS DIGITAL
FAKULTAS EKONOMI DAN BISNIS
UNIVERSITAS KUNINGAN

MODUL PRAKTIKUM
MACHINE LEARNING & AI

DISUSUN OLEH:

Adi Muhamad Muhsidi, S.SI., M.Kom.

NAMA MAHASISWA	: .....
NIM	: .....
KELAS	: .....
TINGKAT / SEMESTER	: .....

PROGRAM STUDI BISNIS DIGITAL

FAKULTAS EKONOMI DAN BISNIS

UNIVERSITAS KUNINGAN

2025

KATA PENGANTAR

Assalamualaikum Warahmatullahi Wabarakatuh,

Puji syukur penulis panjatkan ke hadirat Allah SWT, karena berkat rahmat dan karunia-Nya modul Praktikum Machine Learning & Artificial Intelligence (ML & AI) ini dapat disusun. Modul ini dirancang sebagai panduan praktis bagi mahasiswa Program Studi Bisnis Digital dan Manajemen dalam memahami konsep dasar, teknik, serta implementasi Machine Learning dan Artificial Intelligence, khususnya dalam konteks bisnis digital.

Perkembangan teknologi digital yang semakin pesat menuntut generasi muda, khususnya mahasiswa, untuk tidak hanya menjadi pengguna teknologi, tetapi juga mampu memahami, mengolah, dan mengembangkan teknologi tersebut. Machine Learning dan AI saat ini telah menjadi bagian penting dalam kehidupan sehari-hari, mulai dari sistem rekomendasi, analisis sentimen, deteksi fraud, hingga penerapan chatbot cerdas.

Modul ini disusun dengan pendekatan teori dan praktik yang saling melengkapi. Materi dibagi menjadi beberapa bab yang dimulai dari konsep dasar, teori pendukung, hingga penerapan algoritma ML & AI dengan berbagai studi kasus nyata. Praktikum yang disertakan di dalam modul ini menggunakan Google Colab sebagai platform utama, sehingga memudahkan mahasiswa untuk berlatih tanpa memerlukan instalasi yang kompleks.

Ucapan terima kasih penulis sampaikan kepada seluruh pihak yang telah membantu dalam penyusunan modul ini, baik secara langsung maupun tidak langsung. Semoga modul ini dapat bermanfaat dan menjadi bekal berharga bagi mahasiswa dalam memahami dan mengaplikasikan teknologi AI dan Machine Learning, khususnya untuk menjawab tantangan dunia bisnis digital.

Wassalamualaikum Warahmatullahi Wabarakatuh,

Kuningan, 28 Oktober 2025
Hormat Kami

Penulis

DAFTAR ISI

MODUL I
PENGANTAR ML & AI

A. CAPAIAN PEMBELAJARAN

1) Mahasiswa memahami hierarki konsep AI → ML → DL.
2) Mahasiswa mengetahui klasifikasi AI berdasarkan tingkat kemampuan.
3) Mahasiswa mengenal perjalanan sejarah AI dari masa ke masa.
4) Mahasiswa memahami pentingnya data sebagai “bahan bakar” AI.
5) Mahasiswa mengetahui contoh aplikasi nyata AI dalam kehidupan sehari-hari.

B. DASAR TEORI

1. Definisi dan Hierarki Konsep

Definisi dan Hierarki Konsep Kecerdasan buatan (Artificial Intelligence/AI) merupakan cabang ilmu komputer yang bertujuan menciptakan sistem yang dapat meniru cara berpikir dan belajar manusia. Teknologi ini dirancang agar mampu mengambil keputusan, memahami konteks, serta beradaptasi terhadap situasi yang berubah. Contoh penerapannya beragam, mulai dari aplikasi musik hingga kendaraan otonom. AI tidak sekadar menjalankan instruksi, tetapi juga menganalisis pola dan memprediksi perilaku pengguna.

Salah satu bagian utama dari AI adalah Machine Learning (ML), yaitu pendekatan yang memungkinkan sistem belajar dari data tanpa perlu instruksi eksplisit. Berbeda dari pemrograman konvensional yang bersifat instruktif, ML mengandalkan data historis untuk menemukan pola. Sebagai ilustrasi, dengan data pembelian pelanggan, sistem ML dapat memprediksi kemungkinan pelanggan berhenti berbelanja.

Selanjutnya, terdapat Deep Learning (DL) sebagai subbidang dari ML yang memanfaatkan jaringan saraf tiruan untuk mengolah data besar dan kompleks. Teknologi ini sangat efektif dalam pengenalan wajah, pemrosesan bahasa alami, dan sistem chatbot. Kelebihan DL terletak pada kemampuannya mengenali fitur-fitur mendalam secara otomatis, yang tidak dapat dilakukan metode ML biasa. Untuk memvisualisasikan relasi antar konsep ini, bayangkan tiga lingkaran konsentris: lingkaran terbesar adalah AI, di dalamnya ML, dan paling dalam adalah DL. Struktur ini menunjukkan bahwa DL adalah bagian dari ML, dan ML merupakan bagian dari AI secara keseluruhan.

2. Klasifikasi AI Berdasarkan Kemampuan

Klasifikasi AI Berdasarkan Kemampuan AI dapat dikategorikan berdasarkan tingkat kecerdasannya. Klasifikasi ini membedakan sejauh mana sebuah sistem mampu meniru kecerdasan manusia, yakni: Narrow AI, General AI, dan Super AI.

a) Narrow AI (Weak AI) Jenis AI ini hanya dirancang untuk menyelesaikan tugas-tugas tertentu, seperti sistem rekomendasi produk atau aplikasi navigasi. Meskipun akurat dalam ruang lingkupnya, ia tidak dapat melakukan hal di luar batas tersebut.
b) General AI (Strong AI) General AI adalah konsep kecerdasan buatan yang memiliki fleksibilitas dan kemampuan belajar seperti manusia. Ia mampu menyelesaikan beragam tugas dan memahami konteks berbeda. Namun, hingga kini, teknologi ini masih berada pada tahap penelitian.
c) Super AI Super AI merupakan gambaran masa depan di mana AI mampu melampaui kecerdasan manusia. Meskipun belum terwujud, diskusi mengenai etika dan regulasi AI tingkat lanjut ini sudah berlangsung, mengingat potensi dampaknya terhadap kehidupan manusia.

3. Perjalanan Sejarah AI

Perjalanan Sejarah AI AI berkembang melalui tahapan penting sepanjang sejarahnya, dari sekadar gagasan menjadi teknologi nyata yang digunakan saat ini.

a) Era Pionir (1950–1960) Periode ini ditandai dengan munculnya pemikiran Alan Turing dan pengenalan Turing Test. Tahun 1956, konferensi Dartmouth secara resmi memulai studi AI sebagai bidang ilmiah.
b) AI Winter (1970–1980) Pada fase ini, ekspektasi terhadap AI tidak sejalan dengan kemampuan teknologi saat itu. Keterbatasan perangkat keras dan algoritma menyebabkan kegagalan banyak proyek, yang berdampak pada penurunan minat dan pendanaan.
c) Kebangkitan Modern (1990–sekarang) AI kembali berkembang berkat ketersediaan Big Data, kemajuan GPU, dan kemudahan akses informasi lewat internet. Kini, AI diaplikasikan secara luas dalam berbagai sektor, dari e-commerce hingga kendaraan otonom.

4. Data sebagai Bahan Bakar AI

Data sebagai Bahan Bakar AI Dalam pengembangan AI, data berperan sebagai elemen kunci yang menentukan efektivitas model. Tanpa data, algoritma tidak dapat belajar dan berfungsi. AI memproses dua jenis data utama: data terstruktur (seperti angka dan tabel) dan data tidak terstruktur (seperti gambar, teks, dan suara). Deep Learning sangat berperan dalam memahami data tidak terstruktur yang semakin mendominasi di era digital. Siklus hidup data dalam pengembangan AI meliputi:

a) Pengumpulan: data dikumpulkan dari berbagai sumber seperti sensor atau media sosial.
b) Pembersihan: data dibersihkan dari kesalahan dan duplikasi.
c) Penyimpanan: data yang sudah rapi disimpan di basis data atau cloud.
d) Analisis: data dianalisis untuk membangun dan melatih model AI.

Manajemen data yang baik menjadi syarat utama untuk menghasilkan sistem AI yang andal dan akurat.

5. Aplikasi AI dalam Kehidupan Sehari-hari

Aplikasi AI dalam Kehidupan Sehari-hari AI kini hadir dalam berbagai aspek kehidupan dan mendukung efisiensi serta kenyamanan pengguna.

a) Sistem Rekomendasi Platform digital seperti Netflix, Spotify, dan e-commerce memanfaatkan AI untuk memberikan rekomendasi personal berdasarkan preferensi pengguna.
b) Asisten Virtual & Chatbot Layanan seperti Siri dan Google Assistant membantu pengguna menjalankan perintah suara, sedangkan chatbot digunakan dalam layanan pelanggan untuk merespons pertanyaan secara otomatis.
c) Deteksi Fraud & Keamanan AI mendeteksi aktivitas mencurigakan dalam sistem perbankan, seperti transaksi tak biasa yang langsung memicu sistem perlindungan otomatis.
d) Kendaraan Otonom & Kota Cerdas Mobil otonom menggunakan sensor dan AI untuk mengambil keputusan berkendara secara real-time. Dalam skala kota, AI membantu manajemen lalu lintas dan efisiensi energi.

Dengan berbagai penerapannya, AI telah menjadi bagian integral dalam kehidupan modern. Tantangan ke depan adalah mengelola teknologi ini secara etis agar tetap memberikan manfaat tanpa menimbulkan risiko sosial.

C. LATIHAN DAN TUGAS

1. Latihan

Untuk memperdalam pemahaman mengenai konsep dasar Artificial Intelligence (AI) dan Machine Learning (ML), mahasiswa diminta untuk mengerjakan latihan berikut:

a) Buatlah sebuah *mindmap* sederhana yang menggambarkan hubungan hierarki antara AI → ML → DL. Sertakan definisi singkat pada setiap bagian serta contoh aplikasinya. Mindmap dapat dibuat menggunakan kertas dan alat tulis, atau dengan aplikasi digital seperti Canva, MindMeister, atau Microsoft PowerPoint.

b) Lakukan diskusi kelompok dengan topik: *"Apakah Super AI layak untuk dikhawatirkan?"*. Setiap kelompok diminta memberikan argumen yang mendukung atau menolak pernyataan tersebut, dengan mengaitkannya pada dampak sosial, etika, dan teknologi. Hasil diskusi dapat dipresentasikan secara singkat di depan kelas.

c) Carilah satu contoh aplikasi AI yang sering digunakan dalam kehidupan sehari-hari (misalnya sistem rekomendasi di Netflix, asisten virtual Siri, atau deteksi penipuan di perbankan). Jelaskan secara singkat bagaimana aplikasi tersebut bekerja serta manfaat yang diberikan bagi pengguna.

2. Tugas

Konsep Dasar & Hierarki AI

a) Jelaskan pengertian Artificial Intelligence (AI) menurut pemahaman Anda sendiri. Sebutkan pula tujuan utama dari pengembangan AI.
b) Jelaskan hubungan antara AI, Machine Learning, dan Deep Learning dalam bentuk naratif dan diagram hierarki sederhana.
c) Mengapa Deep Learning semakin populer di era Big Data? Sebutkan minimal 2 alasan utama.
d) Menurut Anda, apa kelebihan dan keterbatasan dari pendekatan "komputer belajar dari data" dibandingkan pendekatan tradisional berbasis aturan (*rule-based*)?

Klasifikasi & Perkembangan AI

a) Jelaskan perbedaan mendasar antara Narrow AI, General AI, dan Super AI dari segi kemampuan dan ruang lingkup.
b) Menurut Anda, apakah Super AI layak untuk dikhawatirkan? Jelaskan alasan Anda dari sisi etika, sosial, dan teknologi.
c) Buatlah tabel perbandingan yang menampilkan:
    a. Kemampuan utama
    b. Contoh aplikasi nyata (jika ada)
    c. Status pengembangan saat ini dari ketiga jenis AI tersebut.
d) Jelaskan secara singkat 3 era penting dalam sejarah perkembangan AI:
    a. Era pionir (1950–1960)
    b. AI Winter (1970–1980)
    c. Kebangkitan modern (1990–sekarang)

Data Sebagai Bahan Bakar AI

a) Jelaskan mengapa data disebut sebagai "bahan bakar" dalam pengembangan AI.
b) Sebutkan perbedaan antara data terstruktur dan tidak terstruktur, sertakan masing-masing 3 contoh nyata.
c) Jelaskan siklus hidup data secara berurutan dari pengumpulan hingga analisis, dan sebutkan apa risiko jika salah satu tahap diabaikan.

d) Jelaskan makna dari ungkapan “garbage in, garbage out” dalam konteks Machine Learning.

Aplikasi & Dampak AI

a) Sebutkan dan jelaskan 4 contoh penerapan AI dalam kehidupan sehari-hari, masing-masing minimal 2 kalimat.
b) Pilih salah satu aplikasi AI yang sering Anda gunakan. Jelaskan bagaimana cara kerja sederhananya dan manfaat apa yang Anda rasakan sebagai pengguna.
c) Menurut Anda, apa dampak positif dan negatif AI terhadap dunia kerja dan pendidikan di masa depan? Jelaskan pendapat Anda secara kritis.

FAKULTAS EKONOMI
UNIVERSITAS KUNINGAN

MODUL II
PARADIGMA MACHINE LEARNING & DATA

A. CAPAIAN PEMBELAJARAN

1) Mahasiswa memahami tiga paradigma utama ML.
2) Mahasiswa memahami dasar statistik yang mendukung analisis data.
3) Mahasiswa mengenal prinsip EDA (Exploratory Data Analysis).
4) Mahasiswa memahami dasar probabilitas dalam konteks data.
5) Mahasiswa mengetahui prinsip storytelling dengan data.

B. DASAR TEORI

1. Paradigma & Jenis Machine Learning

Paradigma & Jenis Machine Learning Machine Learning (ML) adalah bagian dari kecerdasan buatan yang memungkinkan sistem belajar secara otomatis dari data tanpa perlu instruksi eksplisit. Alih-alih menggunakan aturan tetap, ML membangun model berdasarkan pola dalam data. Dalam penerapannya, terdapat tiga pendekatan utama dalam ML, yaitu Supervised Learning, Unsupervised Learning, dan Reinforcement Learning. Ketiganya dibedakan berdasarkan bagaimana model memperoleh pengetahuan dari data yang tersedia.

a) Supervised Learning

Pendekatan ini menggunakan data berlabel, yaitu data yang telah dilengkapi dengan jawaban atau target. Model dilatih untuk mempelajari hubungan antara input (fitur) dan output (label), kemudian digunakan untuk memprediksi hasil dari data baru. Sebagai contoh, untuk memprediksi harga rumah, model menggunakan data seperti luas rumah, lokasi, dan jumlah kamar sebagai input, serta harga sebagai output. Setelah belajar dari data tersebut, model dapat memperkirakan harga rumah lain dengan karakteristik serupa. Supervised learning sangat efektif dalam tugas klasifikasi (misalnya, deteksi email spam) dan regresi (seperti prediksi harga atau permintaan pasar).

b) Unsupervised Learning

unsupervised learning bekerja dengan data tanpa label. Model berusaha menemukan pola tersembunyi atau struktur alami dari data. Contohnya adalah segmentasi pelanggan dalam dunia bisnis, di mana data seperti frekuensi belanja dan nilai transaksi dianalisis untuk mengelompokkan pelanggan ke dalam segmen tertentu, tanpa informasi kelompok sebelumnya. Algoritma seperti K-Means atau PCA sering digunakan dalam konteks ini. Pendekatan ini banyak digunakan untuk clustering, deteksi anomali, dan reduksi dimensi data kompleks.

c) Reinforcement Learning

Paradigma ini berfokus pada proses belajar melalui interaksi dengan lingkungan. Agen akan mencoba berbagai tindakan dan menerima umpan balik berupa reward atau punishment. Tujuan utama adalah memilih tindakan optimal yang menghasilkan hasil terbaik dalam jangka panjang. Contoh populernya adalah AlphaGo, yang mempelajari strategi permainan Go dengan bermain jutaan kali melawan dirinya sendiri. Pendekatan ini juga digunakan dalam robotika dan sistem kontrol otonom. Kelebihannya terletak pada kemampuan adaptasi terhadap kondisi dinamis, meskipun memerlukan proses pelatihan yang intensif dan memakan sumber daya

2. Statistik Deskriptif untuk Data Science

Dalam proses membangun model machine learning, memahami data adalah langkah pertama yang sangat penting. Salah satu cara untuk memahami data adalah dengan menggunakan **statistik deskriptif**, yaitu cabang statistik yang berfungsi untuk menggambarkan dan merangkum data agar lebih mudah dipahami. Statistik deskriptif membantu kita melihat pola umum, tren, maupun penyimpangan dalam data sebelum melangkah ke tahap analisis yang lebih kompleks. Ada beberapa konsep utama dalam statistik deskriptif yang perlu dipahami oleh calon praktisi data science:

```
Drive already mounted at /content/drive;
```

	Luas (m2)	Kamar	Harga (juta)
0	45	2	320
1	60	3	450
2	36	2	250
3	50	2	360
4	70	3	520
5	90	4	700
6	42	2	300
7	55	2	400
8	80	3	610
9	100	4	800
10	65	3	500
11	48	2	330
12	77	3	590
13	85	4	670
14	52	2	370
15	95	4	760
16	60	3	460
17	72	3	540
18	40	2	280
19	88	4	710

a) Measures of Central Tendency (Ukuran Pemusatan Data)

Ukuran pemusatan data menunjukkan nilai tengah atau nilai yang paling mewakili sekumpulan data. Ada tiga ukuran utama yang sering digunakan:

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 20 entries, 0 to 19
Data columns (total 3 columns):
 #   Column        Non-Null Count  Dtype
---  ------        --------------  -----
 0   Luas (m2)     20 non-null     int64
 1   Kamar         20 non-null     int64
 2   Harga (juta)  20 non-null     int64
dtypes: int64(3)
memory usage: 612.0 bytes
```

- **Mean (rata-rata)**: diperoleh dengan menjumlahkan seluruh nilai kemudian membaginya dengan jumlah data. Mean sangat umum digunakan, tetapi sensitif terhadap nilai ekstrem (outlier).
- **Median (nilai tengah)**: nilai yang berada tepat di tengah setelah data diurutkan. Median tidak terpengaruh oleh outlier, sehingga lebih mewakili pusat data jika ada nilai ekstrem.
- **Modus (nilai yang paling sering muncul)**: menunjukkan nilai yang paling banyak muncul dalam kumpulan data. Modus berguna untuk data kategorik atau diskrit.

Contoh Aplikasi di Google Colab

```
df.describe() # digunakan untuk Menampilkan statistik deskriptif untuk kolom numerik (jumlah, rata-rata, standar deviasi, min, max, kuartil).
```

	Luas (m2)	Kamar	Harga (juta)
count	20.000000	20.000000	20.00000
mean	65.500000	2.850000	496.00000
std	19.661611	0.812728	170.95398
min	36.000000	2.000000	250.00000
25%	49.500000	2.000000	352.50000
50%	62.500000	3.000000	480.00000
75%	81.250000	3.250000	625.00000
max	100.000000	4.000000	800.00000

Interpretasi Output

- **Mean Luas: 65.5** Artinya: Rata-rata luas rumah dalam dataset Anda adalah 65.5 meter persegi. Ini adalah ukuran pusat yang memberikan gambaran umum tentang seberapa besar rumah yang ada.
- **Median Luas: 62.5** Artinya: Jika semua rumah diurutkan berdasarkan luasnya, rumah yang berada tepat di tengah memiliki luas 62.5 meter persegi. Nilai median ini sedikit lebih rendah dari mean, yang menunjukkan kemungkinan ada beberapa rumah dengan luas sangat besar yang sedikit "menarik" nilai rata-rata ke atas.
- **Modus Kamar: 2** Artinya: Jumlah kamar yang paling sering muncul atau paling umum di dataset Anda adalah 2 kamar. Ini adalah informasi yang sangat praktis untuk mengetahui tipe rumah yang paling banyak dijual.

- **Mean Harga: 496.0** Artinya: Harga rata-rata rumah di dataset Anda adalah 496 juta Rupiah.
- **Range Harga: 550** Artinya: Selisih harga antara rumah yang paling mahal dan yang paling murah adalah 550 juta Rupiah. Ini menunjukkan seberapa besar rentang harga yang dicakup oleh data Anda.
- **Varians Harga: 29225.26** Artinya: Ini adalah ukuran statistik tentang seberapa jauh setiap data harga tersebar dari rata-ratanya. Angka ini sulit diinterpretasikan secara langsung karena satuannya adalah "(juta Rupiah) kuadrat", namun ini adalah dasar untuk menghitung standar deviasi.
- **Standar Deviasi Harga: 170.95** Artinya: Ini adalah nilai yang paling berguna untuk memahami sebaran harga. Secara rata-rata, harga sebuah rumah di dataset ini menyimpang sekitar 170.95 juta Rupiah dari harga rata-rata (496 juta). Dengan kata lain, sebagian besar rumah kemungkinan memiliki harga di antara (496 - 170.95) juta hingga (496 + 170.95) juta. Standar deviasi yang cukup besar ini menunjukkan bahwa variasi harga rumah di dataset Anda cukup signifikan.

b) Measures of Spread (Ukuran Penyebaran Data)

Selain mengetahui pusat data, penting juga memahami seberapa tersebar data di sekitar pusat tersebut. Ukuran penyebaran membantu menilai homogenitas atau keragaman data. Beberapa ukuran utama adalah:

- **Range (jangkauan)**: selisih antara nilai terbesar dan terkecil.
- **Varians**: rata-rata dari kuadrat selisih tiap nilai terhadap mean. Varians menunjukkan seberapa besar penyebaran data secara keseluruhan.
- **Standar deviasi**: akar kuadrat dari varians. Ukuran ini sering dipakai karena berada pada satuan yang sama dengan data aslinya, sehingga lebih mudah diinterpretasikan.

c) Percentiles & Quartiles (Posisi Data dalam Distribusi)

Percentile menunjukkan posisi relatif suatu nilai dalam distribusi data. Misalnya, jika seorang siswa berada pada **persentil ke-90**, artinya ia memiliki nilai lebih tinggi daripada 90% siswa lainnya. Quartiles membagi data menjadi empat bagian yang sama besar. **Q1** adalah nilai di bawah 25% data, **Q2 (median)** berada di tengah (50%), dan **Q3** di bawah 75% data. Quartiles sering digunakan untuk membuat box plot dan mendeteksi outlier.

d) Skewness & Kurtosis (Bentuk Distribusi Data)

Selain posisi dan penyebaran, bentuk distribusi data juga penting dipahami.

- **Skewness (kemencengan)** mengukur seberapa simetris data terhadap mean. Jika skewness positif, data condong ke kanan (banyak nilai kecil, sedikit nilai besar); jika negatif, condong ke kiri.
- **Kurtosis** mengukur seberapa “runcing” puncak distribusi dibanding distribusi normal. Kurtosis tinggi menunjukkan puncak yang tajam dan banyak nilai ekstrem, sedangkan kurtosis rendah menunjukkan puncak yang landai.

Memahami statistik deskriptif memungkinkan kita mengenali karakteristik dasar data sebelum membangun model machine learning. Tanpa pemahaman ini, kita berisiko salah menafsirkan pola, mengabaikan outlier penting, atau menggunakan metode analisis yang tidak tepat. Dengan kata lain, **statistik deskriptif adalah fondasi dari setiap proses analisis data yang andal**.

3. Memahami Distribusi Data

Setelah mengetahui ukuran pemusatan dan penyebaran data, langkah penting selanjutnya adalah memahami bagaimana data didistribusikan. Distribusi mencerminkan

penyebaran nilai dalam suatu dataset, yang menjadi dasar untuk memilih metode analisis yang sesuai. Dalam data science dan machine learning, distribusi data memengaruhi efektivitas algoritma karena banyak teknik statistik mengandalkan asumsi tertentu mengenai bentuk distribusi.

a) Distribusi Normal Distribusi normal atau distribusi Gauss memiliki bentuk simetris menyerupai lonceng, di mana sebagian besar data terpusat di sekitar nilai rata-rata (mean). Distribusi ini banyak dijumpai pada fenomena alami dan sosial, seperti tinggi badan atau nilai ujian. Karena karakteristiknya yang stabil, banyak metode statistik seperti uji hipotesis dan regresi mengasumsikan data berdistribusi normal.
b) Central Limit Theorem (CLT) CLT menjelaskan bahwa rata-rata dari banyak sampel acak dari suatu populasi akan membentuk distribusi mendekati normal, terlepas dari distribusi awal populasi tersebut. Inilah alasan mengapa mean sering digunakan sebagai representasi utama, dan mengapa distribusi normal menjadi dasar dari berbagai teknik statistik modern. Distribusi Lain (Uniform, Binomial, dsb.) Selain distribusi normal, terdapat distribusi lain yang berguna dalam konteks tertentu:
    - Distribusi Uniform: memberikan peluang sama besar untuk semua nilai dalam rentang tertentu, misalnya pada lemparan dadu
    - Distribusi Binomial: digunakan pada data diskrit dengan dua kemungkinan hasil, seperti sukses atau gagal.
    - Distribusi Poisson: memodelkan jumlah kejadian dalam interval waktu, seperti jumlah pelanggan per jam.
    - Distribusi Eksponensial: menggambarkan waktu antar kejadian, misalnya waktu tunggu kedatangan kendaraan.

Memahami berbagai bentuk distribusi membantu memilih teknik analisis yang tepat dan menafsirkan hasil dengan akurat. Jika data tidak berdistribusi normal, diperlukan pendekatan lain seperti transformasi atau metode non-parametrik.

4. Exploratory Data Analysis (EDA)

Sebelum memulai proses pemodelan machine learning, penting untuk memahami karakteristik data secara menyeluruh melalui tahapan Exploratory Data Analysis (EDA). EDA merupakan fase awal dalam alur kerja data science yang bertujuan mengeksplorasi struktur data, mengidentifikasi pola, mendeteksi anomali, dan menggali hubungan antar variabel. Fokus EDA bukan pada pembuktian statistik, melainkan membangun pemahaman intuitif yang menjadi dasar strategi analisis dan pemodelan selanjutnya.

Dengan melakukan EDA, praktisi data dapat menentukan teknik praproses yang sesuai, memilih algoritma yang tepat, dan menghindari kesalahan umum seperti penggunaan fitur yang tidak relevan atau bias data. Umumnya, EDA dilakukan melalui tiga pendekatan utama:

a) Univariate Analysis: Analisis ini fokus pada satu variabel dalam dataset untuk memahami distribusinya. Teknik yang digunakan antara lain:
- Histogram – menggambarkan distribusi numerik dengan batang frekuensi.
- Bar Chart – menunjukkan frekuensi antar kategori untuk data kategorikal.

Melalui pendekatan ini, kita dapat mengidentifikasi outlier, sebaran data, dan kategori yang tidak seimbang.

b) Bivariate Analysis: Pendekatan ini mengevaluasi hubungan antara dua variabel, baik numerik maupun kombinasi numerik dan kategorikal. Contoh visualisasi yang umum digunakan:

- Scatter Plot – menunjukkan korelasi antara dua variabel numerik.
- Box Plot – membandingkan distribusi numerik berdasarkan kategori tertentu.

Analisis ini membantu mendeteksi hubungan yang signifikan antar fitur dan keterkaitannya dengan target prediksi.

c) Multivariate Analysis: dilakukan untuk memahami interaksi antar lebih dari dua variabel secara simultan. Teknik yang sering digunakan meliputi:

- Correlation Matrix – mengukur kekuatan hubungan antar pasangan fitur.
- Heatmap – memvisualisasikan korelasi dengan gradasi warna.
- Pair Plot – menampilkan scatter plot seluruh pasangan fitur dalam satu grid.

Melalui pendekatan ini, kita dapat mengidentifikasi fitur yang saling tumpang tindih maupun saling melengkapi, sehingga membantu proses seleksi fitur secara lebih efektif.

Secara keseluruhan, EDA adalah fondasi penting dalam proses analisis data. Tanpa EDA, proses modelling ibarat menavigasi tanpa peta—berisiko tinggi terhadap kesalahan interpretasi data. EDA yang menyeluruh memungkinkan pengambilan keputusan yang lebih akurat dan berbasis pemahaman yang kuat terhadap data.

5. Probabilitas dalam Konteks Data Science

Probabilitas merupakan landasan penting dalam dunia data science dan machine learning. Banyak algoritma, terutama yang berkaitan dengan klasifikasi, inferensi, dan evaluasi model, dibangun atas konsep peluang. Dengan pemahaman probabilitas, praktisi data dapat mengelola ketidakpastian, melakukan prediksi, serta menilai tingkat kepercayaan terhadap hasil analisis.

a) Konsep Dasar Peluang

Peluang mengukur seberapa besar kemungkinan suatu kejadian terjadi, dinyatakan dalam rentang antara 0 hingga 1. Nilai **0** berarti kejadian mustahil, sedangkan **1** berarti kejadian pasti terjadi. Rumus sederhana peluang adalah:

$$P(A) = \frac{\text{Jumlah kejadian A}}{\text{Jumlah seluruh kejadian yang mungkin}}$$

Misalnya, jika kita melempar koin, peluang munculnya sisi gambar adalah 1/2 atau 0,5. Dalam data science, konsep dasar ini digunakan misalnya untuk menghitung peluang seorang pelanggan melakukan pembelian, atau peluang sebuah email dikategorikan sebagai spam.

b) Conditional Probability (Peluang Bersyarat)

Conditional probability adalah peluang terjadinya suatu kejadian **dengan syarat bahwa kejadian lain sudah terjadi**. Ini penting karena banyak peristiwa dalam data tidak berdiri sendiri, melainkan saling berkaitan.Rumusnya adalah:

$$P(A \mid B) = \frac{P(A \cap B)}{P(B)}$$

Contoh sederhana: peluang seseorang membeli payung **dengan syarat** hari sedang hujan. Jika dari data diketahui bahwa 80% orang yang membeli payung melakukannya saat hujan, maka kita bisa menuliskan P(Beli Payung | Hujan)=0,8P(\text{Beli Payung | Hujan}) = 0,8P(Beli Payung | Hujan)=0,8. Konsep ini banyak digunakan dalam algoritma seperti Naïve Bayes, yang mendasarkan prediksi pada peluang bersyarat antar fitur dan label.

c) Bayes' Theorem (Teorema Bayes)

Teorema Bayes adalah konsep kunci yang menjelaskan bagaimana kita memperbarui peluang suatu hipotesis berdasarkan bukti baru. Rumusnya adalah:

$$P(H \mid E) = \frac{P(E \mid H) \times P(H)}{P(E)}$$

Dalam bentuk intuitif, Teorema Bayes bekerja seperti ini: kita memiliki keyakinan awal (prior probability) terhadap suatu hal, lalu ketika muncul bukti baru (evidence), kita memperbarui keyakinan tersebut menjadi keyakinan baru (posterior probability).

Contoh sederhana: Misalkan 1% populasi menderita suatu penyakit. Tes diagnostik memiliki akurasi 99%. Jika seseorang hasil tesnya positif, apakah pasti ia sakit? Tidak selalu. Dengan Teorema Bayes, kita menghitung peluang sebenarnya seseorang sakit dengan mempertimbangkan probabilitas penyakit dalam populasi (prior) dan keakuratan tes (likelihood). Hasilnya, peluangnya tidak 99%, melainkan sekitar 50% karena penyakitnya sangat jarang.

Teorema Bayes dan konsep probabilitas bersyarat menjadi dasar dari banyak algoritma machine learning, terutama model berbasis probabilistik seperti Naïve Bayes, Hidden Markov Model, dan Bayesian Network. Memahami konsep ini akan membantu mahasiswa membangun intuisi tentang bagaimana model mengelola ketidakpastian dalam data untuk membuat prediksi yang lebih akurat.

6. Storytelling dengan Data

Data yang akurat belum tentu efektif jika tidak disampaikan dengan cara yang jelas dan mudah dipahami. Di sinilah pentingnya storytelling dalam penyajian data. Storytelling dengan data bukan hanya soal membuat grafik menarik, melainkan juga membangun narasi yang logis dan persuasif berdasarkan bukti data. Dalam praktik data science, kemampuan mengomunikasikan hasil analisis secara visual dan naratif sangat penting agar informasi yang diperoleh bisa dipahami dan ditindaklanjuti oleh para pengambil keputusan.

a) Prinsip Visualisasi yang Efektif Visualisasi yang baik menyederhanakan kompleksitas data dan menyoroti pesan utama. Beberapa prinsip kuncinya:

- Fokus dan sederhana – hindari elemen visual berlebihan, cukup gunakan warna dan label seperlunya.
- Kontekstual – tambahkan judul, keterangan, dan satuan yang relevan untuk memperjelas isi grafik.
- Konsisten – pertahankan gaya visual, warna, dan skala agar mudah dibandingkan.
- Tegas – pastikan pesan utama langsung terlihat tanpa perlu interpretasi rumit.

b) Memilih Grafik Sesuai Tipe Data Pemilihan grafik yang tepat membantu audiens lebih cepat memahami isi data. Beberapa panduan umum:

- Data kategorik → gunakan bar chart atau pie chart.
- Data numerik tunggal → histogram untuk distribusi.
- Dua variabel numerik → scatter plot untuk melihat korelasi.
- Data waktu → line chart untuk tren dari waktu ke waktu
- Data dengan outlier → box plot.
- Banyak variabel → heatmap atau pair plot.

Grafik yang sesuai mencegah kesalahan interpretasi dan memperkuat pesan yang ingin disampaikan.

c) Kesalahan Umum dalam Visualisasi Meskipun terlihat sederhana, visualisasi data sering kali mengandung kesalahan yang menyesatkan. Beberapa kesalahan umum meliputi:

- Memotong sumbu yang membuat perbedaan tampak lebih ekstrem.
- Menggunakan warna atau efek berlebihan yang mengalihkan perhatian.
- Menyajikan data tanpa konteks, seperti tanpa skala atau pembanding.
- Menyalahgunakan proporsi, misalnya pie chart yang tidak utuh.

Kesalahan seperti ini dapat menimbulkan misinterpretasi dan merusak kredibilitas analisis. Oleh karena itu, storytelling dengan data menuntut ketelitian dan kejelasan. Seorang data scientist tidak cukup hanya pintar menganalisis, tetapi juga harus mampu menyampaikan temuannya secara komunikatif. Dengan visualisasi yang tepat, data dapat diubah menjadi cerita yang mendorong aksi.

C. LATIHAN DAN TUGAS

1. Pemahaman Paradigma Machine Learning

a) Jelaskan perbedaan mendasar antara Supervised Learning, Unsupervised Learning, dan Reinforcement Learning dalam hal jenis data masukan, tujuan pembelajaran, dan contoh kasus aplikasinya.
b) Berikan 2 contoh kasus nyata (dunia bisnis atau kehidupan sehari-hari) yang cocok untuk masing-masing paradigma (total 6 contoh).
c) Menurut Anda, mengapa Reinforcement Learning sering digunakan pada bidang robotika atau game? Jelaskan logikanya.
d) Buatlah diagram sederhana yang menggambarkan alur kerja (workflow) dari ketiga paradigma tersebut.

2. Fondasi Data & Statistik

a) Jelaskan perbedaan antara data terstruktur dan data tidak terstruktur, serta berikan masing-masing 3 contoh nyata.
b) Sebutkan dan jelaskan tahapan dalam siklus hidup data mulai dari pengumpulan hingga analisis.
c) Mengapa kualitas data sangat penting dalam membangun model Machine Learning? Jelaskan konsep “garbage in, garbage out”.
d) Jelaskan makna dari ukuran statistik berikut: mean, median, modus, varians, dan standar deviasi — dan beri contoh interpretasinya dalam konteks nilai ujian mahasiswa.

MODUL III
PENGENALAN PYTHON & GOOGLE COLAB

A. CAPAIAN PEMBELAJARAN

1) Mahasiswa mengenal Python sebagai bahasa utama dalam Machine Learning & AI.
2) Mahasiswa memahami alasan penggunaan Python dalam riset dan industri AI.
3) Mahasiswa mampu melakukan setup Google Colab sebagai lingkungan coding.
4) Mahasiswa mengenal cara menjalankan kode, mengelola file, dan menggunakan library dasar.

B. DASAR TEORI

1. Bahasa Pemrograman Python dalam AI

Bahasa Pemrograman Python dalam AI Python menjadi bahasa pemrograman utama dalam pengembangan kecerdasan buatan (AI) dan machine learning (ML) berkat kemudahan penggunaan dan ekosistemnya yang kuat. Dikembangkan oleh Guido van Rossum dan dirilis pada 1991, Python dirancang agar mudah dibaca dan ditulis, bahkan oleh pemula. Nama "Python" sendiri terinspirasi dari grup komedi Monty Python, bukan hewan ular.

Kepopuleran Python dalam AI muncul karena beberapa keunggulan. Pertama, sintaksnya sederhana dan menyerupai bahasa manusia, memudahkan penulisan kode yang cepat dan minim kesalahan. Kedua, komunitasnya besar dan aktif, menyediakan banyak dokumentasi, tutorial, dan forum bantuan. Ketiga, Python didukung oleh berbagai library seperti NumPy, Pandas, Scikit-learn, TensorFlow, dan PyTorch, yang mempercepat proses pengembangan dari tahap awal hingga produksi.

Dibanding bahasa lain, Python unggul dalam kecepatan pengembangan dan fleksibilitas. R dikenal baik dalam analisis statistik, namun kurang cocok untuk produksi skala besar. Java dan C++ unggul dalam performa, tetapi lebih rumit secara sintaks. Dengan Python, peneliti dan pengembang bisa lebih cepat mewujudkan ide menjadi prototipe dan aplikasi. Inilah alasan mengapa Python menjadi keterampilan wajib bagi siapa pun yang ingin menekuni AI.

2. Ekosistem Python untuk AI

Keunggulan Python dalam AI tidak lepas dari ekosistem library-nya yang lengkap. Library ini mencakup seluruh tahapan pengembangan AI, mulai dari pengolahan data hingga deployment model. Untuk pengolahan data, NumPy dan Pandas adalah dua library utama. NumPy mendukung operasi numerik efisien, sementara Pandas memudahkan manipulasi data dalam bentuk tabel (dataframe). Keduanya sangat penting di tahap awal analisis.

Dalam visualisasi data, Python menawarkan Matplotlib dan Seaborn. Matplotlib fleksibel untuk membuat grafik dasar, sedangkan Seaborn menyajikan grafik statistik dengan tampilan lebih menarik, seperti heatmap dan pairplot. Kombinasi keduanya membantu menyampaikan insight data secara visual. Pada tahap pembangunan model, Scikit-learn menyediakan algoritma machine learning klasik, cocok untuk pemula. Untuk model deep learning, TensorFlow dan Keras menjadi pilihan utama, memungkinkan pembuatan neural network skala besar dengan sintaks yang mudah.

Setelah model selesai, Python juga mendukung deployment. Flask dan FastAPI digunakan untuk membangun API yang menghubungkan model dengan pengguna. Streamlit memungkinkan pembuatan dashboard interaktif hanya dengan kode Python sederhana, cocok untuk presentasi hasil analisis ke pengguna non-teknis. Dengan ekosistem menyeluruh, Python mendukung proses pengembangan AI secara end-to-end. Mahasiswa

yang menguasainya akan lebih siap dalam membangun dan mengimplementasikan solusi AI modern.

3. Google Colab sebagai Tools Praktikum

Google Colaboratory (Colab) adalah platform berbasis cloud yang memudahkan mahasiswa belajar dan mengembangkan proyek AI tanpa perlu instalasi perangkat tambahan. Colab memungkinkan pengguna menjalankan kode Python langsung dari browser, lengkap dengan akses ke GPU dan TPU secara gratis. Untuk menggunakannya, cukup login ke akun Google dan buka situs https://colab.research.google.com . Mahasiswa bisa langsung membuat notebook baru dan menulis kode Python tanpa repot mengatur environment. Fitur kolaborasi real-time memungkinkan kerja kelompok yang efisien, mirip Google Docs.

Dibandingkan Jupyter Notebook lokal, Colab lebih praktis untuk pemula karena sudah dilengkapi dengan library populer dan tidak memerlukan instalasi. Integrasinya dengan Google Drive juga memudahkan penyimpanan, akses, dan berbagi file, baik dataset, model, maupun notebook analisis. Notebook Colab terdiri dari dua jenis sel: Code Cell untuk menulis kode dan Markdown Cell untuk menambahkan penjelasan atau dokumentasi. Struktur ini membuat Colab sangat ideal sebagai media pembelajaran interaktif, di mana teori, kode, dan hasil dapat disajikan dalam satu dokumen terintegrasi.

4. Persiapan Praktikum

Sebelum memulai praktik AI, mahasiswa perlu menyiapkan lingkungan kerja. Google Colab adalah pilihan ideal karena siap pakai, mudah digunakan, dan mendukung semua kebutuhan awal praktikum. Langkah awal yaitu membuat notebook baru melalui menu File → New Notebook di situs Colab. Notebook ini otomatis tersimpan di Google Drive dan memiliki format .ipynb. Di dalamnya, mahasiswa bisa menulis kode Python, mengimpor library seperti NumPy atau Pandas, dan menyimpan pekerjaan mereka secara online.

Setelah notebook dibuat, langkah berikutnya adalah **menulis kode Python sederhana untuk mengenal sintaks dasar**. Kode pertama yang lazim digunakan adalah menampilkan teks ke layar dengan perintah print(). Misalnya:

```
print("Hello, AI World!")
```

Mahasiswa juga perlu mengenal **konsep variabel dan tipe data** dalam Python. Variabel digunakan untuk menyimpan nilai seperti teks (string), angka (integer, float), atau logika (boolean). Contoh:

```
nama = "Adi Muhamad"        # string
umur = 21                   # integer
ipk = 3.75                  # float
is_mahasiswa = True         # boolean

print(nama, umur, ipk, is_mahasiswa)
print(type(nama), type(umur), type(ipk), type(is_mahasiswa))
```

Selanjutnya, untuk mempersiapkan proses analisis data dan machine learning, mahasiswa perlu **mengimpor library dasar seperti NumPy, Pandas, dan Matplotlib**. Library ini merupakan fondasi utama hampir semua proyek AI. Contohnya:

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

Setelah menulis beberapa sel kode, notebook dapat **disimpan dan dibagikan**. Klik menu **File → Save a copy in Drive** untuk menyimpan salinan notebook di Google Drive. Untuk

membagikannya kepada dosen, klik tombol **Share** di pojok kanan atas, lalu ubah pengaturan menjadi **Anyone with the link can view**, kemudian salin tautannya.

Langkah-langkah ini mungkin terlihat sederhana, tetapi merupakan **tahapan penting untuk membangun kebiasaan kerja yang rapi dan kolaboratif**. Mahasiswa yang disiplin dalam mengatur notebook sejak awal akan lebih mudah mengelola kode, data, dan hasil analisis pada proyek-proyek selanjutnya.

5. Kumpulan Sintaks Dasar untuk Praktikum AI di Google Colab

Selain memahami teori, mahasiswa juga perlu mengenal berbagai **sintaks (kode Python) yang sering digunakan** dalam pembelajaran machine learning dan artificial intelligence. Berikut adalah kumpulan sintaks dasar yang dikelompokkan berdasarkan fungsinya, yang dapat digunakan sebagai referensi praktikum.

a) Pengaturan & Interaksi Google Colab

Beberapa perintah penting untuk mempermudah pengelolaan file dan interaksi dengan lingkungan Colab:

Sintaks	Fungsi
`from google.colab import drive; drive.mount('/content/drive')`	Menghubungkan Google Colab dengan Google Drive Anda untuk mengakses file.
`!pip install nama_library`	Menginstal library Python yang belum tersedia di Colab.
`!ls`	Melihat daftar file dan folder di direktori saat ini.
`import library as alias`	Mengimpor library Python agar bisa digunakan dalam kode dengan alias yang lebih pendek.

b) Manajemen Data dengan Pandas

Library Pandas digunakan untuk membaca, menulis, dan mengelola data dalam bentuk tabel (DataFrame):

Sintaks	Fungsi
`pd.read_csv('path/ke/file.csv')`	Membaca file CSV dan memuatnya ke dalam sebuah DataFrame Pandas.
`df.head(n)`	Menampilkan n baris pertama dari DataFrame.
`df.tail(n)`	Menampilkan n baris terakhir dari DataFrame.
`df.info()`	Memberikan ringkasan teknis DataFrame termasuk jumlah baris nama kolom tipe data dan penggunaan memori.
`df.describe()`	Menghasilkan statistik deskriptif untuk kolom numerik.
`df.shape`	Menampilkan dimensi DataFrame dalam format (jumlah baris jumlah kolom).
`df['nama_kolom']`	Memilih satu kolom spesifik dari DataFrame.
`df[['kolom1','kolom2']]`	Memilih beberapa kolom spesifik dari DataFrame.
`df.isnull().sum()`	Menghitung jumlah data kosong (missing values) di setiap kolom.
`df.dropna()`	Menghapus baris yang memiliki nilai kosong.
`df.fillna(value)`	Mengisi nilai kosong dengan nilai tertentu.

c) Visualisasi Data dengan Matplotlib & Seaborn

Visualisasi membantu memahami pola data secara intuitif:

Sintaks	Fungsi
`plt.figure(figsize=(x,y))`	Membuat kanvas plot dengan ukuran tertentu.
`plt.scatter(x,y)`	Membuat scatter plot untuk melihat hubungan antara dua variabel numerik.
`plt.hist(data,bins=n)`	Membuat histogram untuk melihat distribusi dari sebuah variabel.
`sns.heatmap(data)`	Membuat heatmap untuk memvisualisasikan matriks korelasi.
`plt.title('Judul Plot')`	Memberi judul pada plot.
`plt.xlabel('Label X');` `plt.ylabel('Label Y')`	Memberi label pada sumbu X dan Y.
`plt.show()`	Menampilkan plot yang telah dibuat.

d) Preprocessing & Membangun Model (Scikit-learn)

Scikit-learn adalah library utama untuk machine learning klasik:

Sintaks	Fungsi
`from sklearn.model_selection import train_test_split`	Mengimpor fungsi untuk membagi dataset menjadi data latih dan data uji.
`X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)`	Membagi fitur (X) dan target (y) menjadi set training (80%) dan testing (20%). random_state untuk memastikan pembagiannya konsisten.
`from sklearn.linear_model import LinearRegression`	Mengimpor kelas model dari library Scikit-learn.
`model = NamaModel()`	Membuat sebuah instance (objek) dari model yang akan digunakan.
`model.fit(X_train,y_train)`	Melatih (training) model menggunakan data latih.
`predictions = model.predict(X_test)`	Membuat prediksi pada data baru (data uji).
`from sklearn.metrics import accuracy_score`	Mengimpor fungsi untuk mengevaluasi kinerja model.
`score = nama_metrik(y_test2,predictions)`	Membandingkan label asli dari data uji dengan hasil prediksi model.

C. PRAKTIKUM

1. Setup Colab & Python Dasar

a) Tujuan Praktikum

- ✓ Membuat notebook baru di Google Colab.
- ✓ Menulis program Python sederhana (Hello World, operasi aritmatika, list/dictionary).
- ✓ Menyimpan file di Google Drive dan **membagikan** ke dosen.

b) Membuat Notebook Baru di Google Colab

1) Buka browser → kunjungi **https://colab.research.google.com** lalu **Login** dengan akun Google.
2) Klik **File** → **New notebook** (Colab akan membuat file .ipynb baru).
3) **Ubah nama** notebook (klik judul di kiri atas) → gunakan format:
4) ML-Bab3_NIM_NamaLengkap.ipynb
5) (Opsional) **Atur GPU**: Runtime → Change runtime type → Hardware accelerator: GPU → Save.

a. Cek GPU: jalankan sel kode ini:
b. !nvidia-smi
c. Jika tidak perlu GPU untuk bab ini, biarkan **None**.

c) Menulis Program Python Sederhana

Fokus: mengenal **print**, **variabel**, **tipe data**, serta **struktur data** (list & dictionary).

1) "Hello World"

Di **Code Cell**, ketik lalu **Run** (tombol ▶ atau Shift+Enter):

```
print("Hello, AI World!")
```

```
Hello, AI World!
```

2) Variabel & Tipe Data

```
nama = "Adi Muhamad"        # string
umur = 21                   # integer
ipk = 3.75                  # float
is_mahasiswa = True         # boolean

print(nama, umur, ipk, is_mahasiswa)
print(type(nama), type(umur), type(ipk), type(is_mahasiswa))
```

```
Adi Muhamad 21 3.75 True
<class 'str'> <class 'int'> <class 'float'> <class 'bool'>
```

3) Operasi Aritmatika Dasar

```
a, b = 12, 5
print("Tambah:", a + b)
print("Kurang:", a - b)
print("Kali:", a * b)
print("Bagi:", a / b)
print("Modulus:", a % b)
print("Pangkat:", a ** 2)
```

```
Tambah: 17
Kurang: 7
Kali: 60
Bagi: 2.4
Modulus: 2
Pangkat: 144
```

4) List & Dictionary (Struktur Data Penting)

```
# List: urutan data yang bisa diubah
nilai = [80, 85, 90, 88]
nilai.append(92)            # tambah elemen
print("Rata-rata:", sum(nilai)/len(nilai))

# Dictionary: pasangan kunci-nilai
biodata = {
    "nama": "Adi Muhamad",
    "prodi": "Bisnis Digital",
    "minat": ["AI", "Big Data"]
}
print(biodata["nama"], "minat:", ", ".join(biodata["minat"]))
```

```
Rata-rata: 87.0
Adi Muhamad minat: AI, Big Data
```

Tips: Jalankan setiap sel **satu per satu** agar mudah menelusuri error.

d) Import Library Dasar (NumPy, Pandas, Matplotlib)

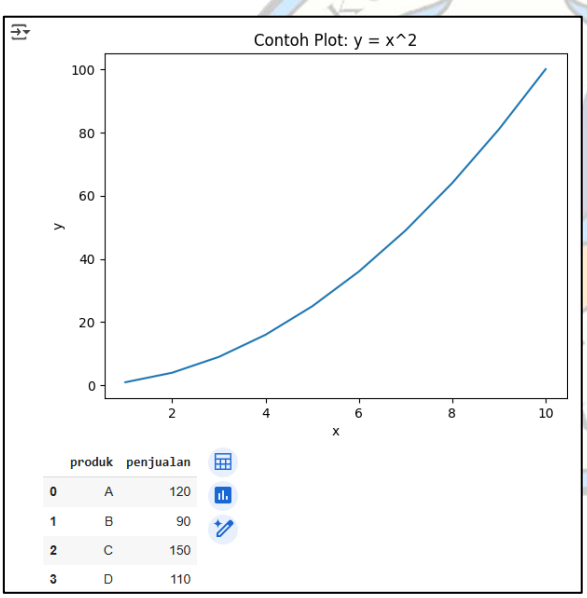
Tujuan: menyiapkan alat minimal untuk analisis data.

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

# Contoh mini: buat data sederhana & plot
x = np.arange(1, 11)
y = x ** 2
plt.plot(x, y)
plt.title("Contoh Plot: y = x^2")
plt.xlabel("x"); plt.ylabel("y")
plt.show()

# Contoh DataFrame ringkas
df = pd.DataFrame({
    "produk": ["A","B","C","D"],
    "penjualan": [120, 90, 150, 110]
})
df
```

Jika suatu library tidak tersedia (jarang terjadi di Colab), Anda bisa install:

```
!pip install seaborn
```

Output dan hasil akhirnya bisa di lihat seperti gambar di bawah ini:



	produk	penjualan
0	A	120
1	B	90
2	C	150
3	D	110

e) Menyimpan ke Google Drive & Membagikan ke Dosen

Simpan ke Drive (otomatis jika Anda login)

1) Klik File → Save a copy in Drive (pastikan muncul ikon Drive di judul).
2) (Opsional) Pindahkan file ke folder khusus (di Google Drive → My Drive/ML-AI/Praktikum).

Bagikan Tautan ke Dosen

1) Klik tombol Share (kanan atas).
2) Pada General access → ubah ke Anyone with the link → Viewer (Can view).
3) Klik Copy link → kirim tautan di LMS/grup kelas sesuai instruksi dosen.

Format penamaan & pengiriman:

- Nama file: ML-Bab3_NIM_NamaLengkap.ipynb
- Subjek pengumpulan (jika via email/LMS): [ML Bab 3] NIM - Nama Lengkap
- Sertakan link Colab & status akses “Anyone with the link can view”.

2. Manipulasi Data & Visualisasi Dasar untuk AI/ML di Google Colab

Cakupan Konsep:

- Membuat Array (NumPy)
- Operasi dasar pada Array (indexing, slicing, broadcasting, agregasi)
- Membuat DataFrame (Pandas)
- Operasi dasar pada DataFrame (seleksi, filter, kolom baru, missing values)
- Visualisasi data dengan Matplotlib & Seaborn

Tujuan Praktikum:

✓ Mahasiswa memahami cara membuat dan memanipulasi array NumPy.
✓ Mahasiswa mampu membaca, mengeksplorasi, dan mengolah data tabular dengan Pandas.
✓ Mahasiswa dapat membuat visualisasi dasar untuk eksplorasi data.

a) Setup & Import Library

```
# Cek versi Python (opsional)
!python --version

# Import library inti
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns

# Opsi tampilan
pd.set_option('display.max_columns', None)
pd.set_option('display.precision', 3)

# Atur ukuran default plot
plt.rcParams['figure.figsize'] = (7,4)

print("Library siap digunakan.")
```

```
Python 3.12.11
Library siap digunakan.
```

b) NumPy — Array & Operasi Dasar

- Membuat Array

```
# Array 1D & 2D
a1 = np.array([1, 2, 3, 4, 5])
a2 = np.array([[1, 2, 3], [4, 5, 6]])

print("a1:", a1, "| shape:", a1.shape, "| dtype:", a1.dtype)
print("a2:\n", a2, "\nshape:", a2.shape)

# Array khusus
zeros = np.zeros((2,3))
ones = np.ones((3,2))
randn = np.random.randn(2,3)  # distribusi normal
arange = np.arange(0, 10, 2)  # start, stop, step

print("\nZeros:\n", zeros)
print("\nOnes:\n", ones)
print("\nRandom normal:\n", randn)
print("\nArange 0..10 step 2:", arange)
```

Output:

```
a1: [1 2 3 4 5] | shape: (5,) | dtype: int64
a2:
 [[1 2 3]
 [4 5 6]]
shape: (2, 3)

Zeros:
 [[0. 0. 0.]
 [0. 0. 0.]]

Ones:
 [[1. 1.]
 [1. 1.]
 [1. 1.]]

Random normal:
 [[ 0.0828677  -1.74319966 -1.60490227]
 [ 0.86445279 -0.22917151  0.63012948]]

Arange 0..10 step 2: [0 2 4 6 8]
```

- Indexing, Slicing, Broadcasting, Agregasi

```python
b = np.arange(1, 13).reshape(3,4)
print("b =\n", b)

# Indexing & slicing
print("\nElemen baris-1 kolom-2:", b[0,1])
print("Baris ke-2 (indeks 1):", b[1])
print("Kolom ke-3 (indeks 2):", b[:,2])
print("Sub-matriks (baris 0..1, kolom 1..3):\n", b[0:2, 1:4])

# Broadcasting
c = b + 10
print("\nBroadcasting b + 10:\n", c)

# Agregasi
print("\nSum keseluruhan:", b.sum())
print("Mean per kolom:", b.mean(axis=0))
print("Max per baris:", b.max(axis=1))

# Transformasi bentuk
print("\nReshape (2,6):\n", b.reshape(2,6))
print("Transpose:\n", b.T)
```

c) Pandas — Membuat & Mengelola DataFrame

Kita akan mencoba dua cara: (a) membuat DataFrame dari dictionary, dan (b) membaca dataset CSV (dataset rumah) yang sudah disiapkan dosen.

- DataFrame dari Dictionary (Contoh Kecil)

```python
# DataFrame kecil
df_demo = pd.DataFrame({
    "produk": ["A","B","C","D"],
    "harga": [100, 120, 90, 150],
    "penjualan": [120, 90, 150, 110]
})
df_demo
```

	produk	harga	penjualan
0	A	100	120
1	B	120	90
2	C	90	150
3	D	150	110

- Membaca Dataset CSV (Prediksi Harga Rumah)

```python
# Membaca Dataset CSV (Prediksi Harga Rumah)
# Mount Google Drive
from google.colab import drive
drive.mount('/content/drive')

# Path file dataset (ganti sesuai lokasi pasti di Drive Anda)
path = '/content/drive/MyDrive/ColabNotebooks/Modul3/dataset_prediksi_harga_rumah.csv'

# Baca dataset ke dalam DataFrame
df = pd.read_csv(path)

# Tampilkan 5 baris pertama
df.head()
```

Output:

```
Drive already mounted at /content/drive;
```

	Luas (m2)	Kamar	Harga (juta)
0	45	2	320
1	60	3	450
2	36	2	250
3	50	2	360
4	70	3	520

- Operasi Dasar pada DataFrame

```python
# Info & ringkasan
print("\nInfo:")
print(df.info())
print("\nDescriptive stats:")
display(df.describe())

# Seleksi kolom & baris
kolom_numerik = df.select_dtypes(include=np.number).columns.tolist()
print("\nKolom numerik:", kolom_numerik)

# head/tail
display(df.head(3))
display(df.tail(3))

# Akses baris dengan iloc/loc
print("\nBaris ke-5 (iloc):\n", df.iloc[4])
# Filter sederhana (jika kolom tersedia)
if 'kamar_tidur' in df.columns:
    print("\nFilter kamar_tidur >= 3:")
    display(df[df['kamar_tidur'] >= 3].head())

# Kolom baru (contoh rasio harga per m2 bangunan)
if set(['harga','luas_bangunan']).issubset(df.columns):
    df['harga_per_m2'] = df['harga'] / df['luas_bangunan']
    display(df[['harga','luas_bangunan','harga_per_m2']].head())

# Menangani missing values (contoh sederhana)
print("\nJumlah missing per kolom:")
print(df.isna().sum())
df_clean = df.fillna(df.median(numeric_only=True))
```

Outputnya:

```
Info:
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 20 entries, 0 to 19
Data columns (total 3 columns):
 #   Column        Non-Null Count  Dtype
---  ------        --------------  -----
 0   Luas (m2)     20 non-null     int64
 1   Kamar         20 non-null     int64
 2   Harga (juta)  20 non-null     int64
dtypes: int64(3)
memory usage: 612.0 bytes
None

Descriptive stats:
```

	Luas (m2)	Kamar	Harga (juta)
count	20.000	20.000	20.000
mean	65.500	2.850	496.000
std	19.662	0.813	170.954
min	36.000	2.000	250.000
25%	49.500	2.000	352.500
50%	62.500	3.000	480.000
75%	81.250	3.250	625.000
max	100.000	4.000	800.000

```
Kolom numerik: ['Luas (m2)', 'Kamar', 'Harga (juta)']
```

	Luas (m2)	Kamar	Harga (juta)
0	45	2	320
1	60	3	450
2	36	2	250

	Luas (m2)	Kamar	Harga (juta)
17	72	3	540
18	40	2	280
19	88	4	710

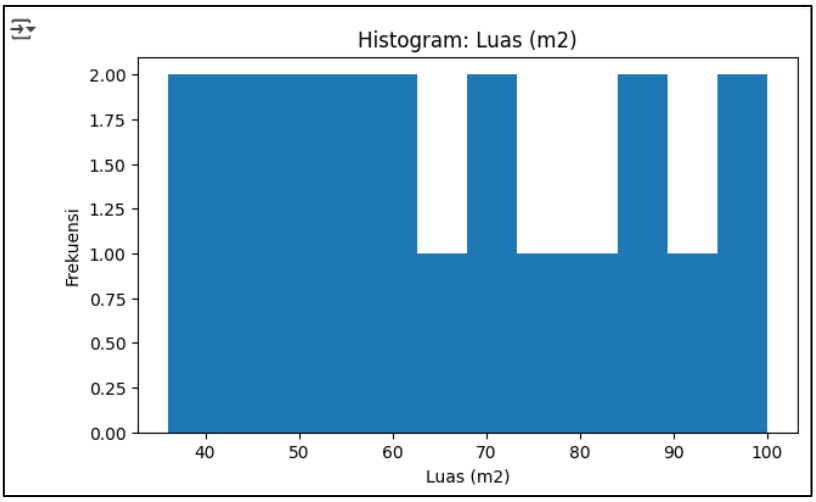
```
Baris ke-5 (iloc):
 Luas (m2)        70
Kamar              3
Harga (juta)     520
Name: 4, dtype: int64

Jumlah missing per kolom:
Luas (m2)        0
Kamar            0
Harga (juta)     0
dtype: int64
```

d) Visualisasi Data — Matplotlib & Seaborn

Kita akan membuat beberapa grafik dasar untuk eksplorasi data.

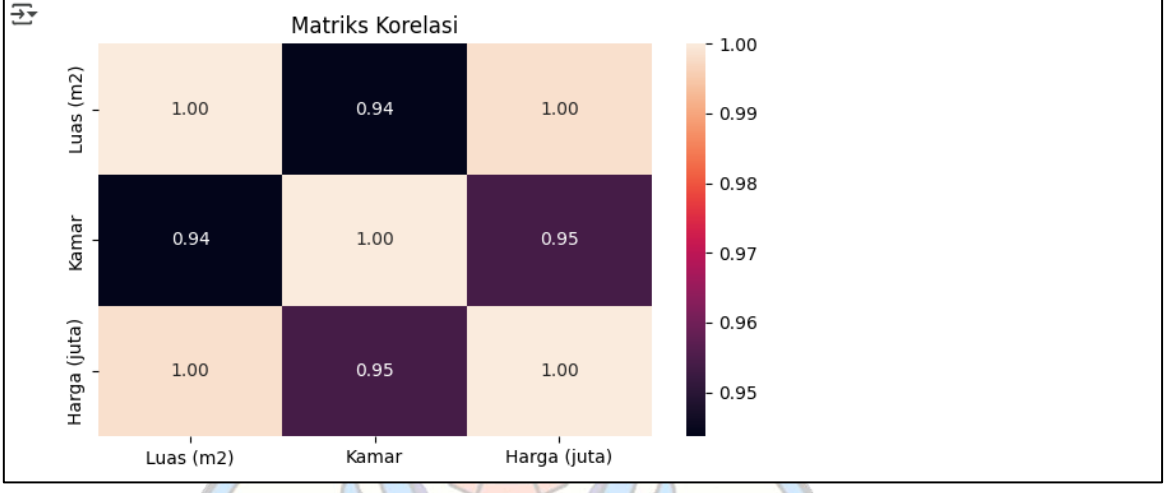
- Histogram salah satu kolom numerik

```python
# Histogram salah satu kolom numerik
kolom = 'luas_bangunan' if 'luas_bangunan' in df.columns else df.select_dtypes(include=np.number).columns[0]
plt.hist(df[kolom], bins=12)
plt.title(f"Histogram: {kolom}")
plt.xlabel(kolom); plt.ylabel("Frekuensi")
plt.show()

# Scatter plot: luas_bangunan vs harga (jika tersedia)
if set(['luas_bangunan','harga']).issubset(df.columns):
    plt.scatter(df['luas_bangunan'], df['harga'])
    plt.title("Scatter: Luas Bangunan vs Harga")
    plt.xlabel("Luas Bangunan (m2)"); plt.ylabel("Harga (Rp)")
    plt.show()
```



- Seaborn: heatmap korelasi & pairplotcatter plot: luas_bangunan vs harga

```
# Seaborn: heatmap korelasi & pairplot
corr = df.select_dtypes(include=np.number).corr(numeric_only=True)
sns.heatmap(corr, annot=True, fmt=".2f")
plt.title("Matriks Korelasi")
plt.show()

kol... = [c for c in ['luas_bangunan','luas_tanah','kamar_tidur','kamar_mandi','harga'] if c in df.columns]
if ...m_pair) >= 3:
    sns.pairplot(df[kolom_pair])
```



D. LATIHAN DAN TUGAS

1. Latihan

Tujuan: Melatih penggunaan print, variabel, tipe data, list, dan dictionary di Colab.

a) Buat notebook baru di Colab dengan nama:
b) ML-Bab3-Individu_NIM_NamaLengkap.ipynb
c) Tulis kode yang menampilkan:
   - Biodata (nama, NIM, prodi, minat) dengan print().
   - Variabel (string, integer/float, boolean) dan tampilkan type()-nya.
   - **List** berisi minimal 5 hobi/minat; tampilkan panjang list dan 1–2 elemen.
   - **Dictionary** berisi biodata ringkas (nama, angkatan, kontak); akses 2 kunci dan tampilkan nilainya.
d) Tambahkan **1 Markdown cell** (judul + deskripsi singkat notebook).
e) Simpan di Google Drive dan **bagikan** (Anyone with the link → Viewer)

2. Tugas

Kerjakan secara individu. Jawab dalam 1 file dokumen (PDF atau langsung di Google Docs). Setiap jawaban minimal 1 paragraf (5–7 kalimat) dan gunakan bahasa ilmiah yang baik. Sertakan identitas di bagian atas (Nama, NIM, Prodi, Kelas). Upload di LMS sesuai petunjuk dosen.

a) Google Colab vs Jupyter Lokal
   - Dalam konteks pembelajaran, apa kelebihan Google Colab dibandingkan coding di laptop lokal (Jupyter/IDE)?
   - Kapan Jupyter Notebook lokal lebih unggul daripada Google Colab?
   - Bagaimana strategi mengatasi ketergantungan internet saat menggunakan Colab?

b) Jelaskan mengapa Python menjadi bahasa pemrograman utama dalam bidang AI dan machine learning. Sertakan minimal tiga alasan utama beserta penjelasannya.

c) Sebutkan dan jelaskan kategori utama library Python untuk AI beserta contoh library dan fungsinya masing-masing (minimal 4 kategori).
d) Jelaskan perbedaan antara Code Cell dan Markdown Cell dalam Google Colab, serta kapan masing-masing sebaiknya digunakan dalam proses praktikum.
e) Anda diminta membuat notebook baru untuk praktikum pertama. Sebutkan langkah-langkah setup awal yang harus dilakukan di Google Colab sebelum mulai menulis kode Python.
f) Berikan contoh nyata aktivitas Anda saat menggunakan Google Colab yang memanfaatkan integrasi dengan Google Drive (misalnya akses dataset atau menyimpan hasil).
g) Menurut Anda, apa tantangan utama belajar pemrograman Python untuk AI bagi pemula, dan strategi apa yang bisa dilakukan untuk mengatasinya?

FAKULTAS EKONOMI
UNIVERSITAS KUNINGAN

MODUL IV
REGRESI LINEAR & PREDIKSI BISNIS

A. CAPAIAN PEMBELAJARAN

1) Mahasiswa memahami konsep regresi linear sederhana & berganda.
2) Mahasiswa mengetahui peran regresi linear dalam analisis bisnis digital.
3) Mahasiswa mampu mengimplementasikan regresi linear menggunakan Scikit-Learn.
4) Mahasiswa mampu mengevaluasi hasil model prediksi.

B. DASAR TEORI

1. Pengertian Fungsi

Pengertian Fungsi Regresi linear adalah teknik dasar dalam machine learning yang digunakan untuk memprediksi variabel target (Y) berdasarkan satu atau lebih variabel input (X). Pendekatan ini membentuk hubungan matematis berupa garis lurus, dan sering dimanfaatkan dalam bisnis untuk memperkirakan penjualan, harga, permintaan, hingga perilaku konsumen.

a) Regresi Linear Sederhana (Simple Linear Regression)

Regresi linear sederhana digunakan ketika hanya ada satu variabel bebas (X) yang digunakan untuk memprediksi satu variabel terikat (Y). Bentuk umum persamaan regresinya adalah:

$$Y = a + bX$$

- Y = nilai target (yang diprediksi),
- X = nilai fitur (variabel independen),
- a = intercept (titik potong sumbu Y),
- b = slope/koefisien regresi (menunjukkan seberapa besar perubahan Y saat X naik satu satuan).

Sebagai contoh, perusahaan menggunakan biaya iklan (X) untuk memprediksi penjualan (Y).

b) Regresi Linear Berganda (Multiple Linear Regression)

Digunakan Jika prediksi melibatkan lebih dari satu variabel bebas. Persamaan umumnya:

$$Y = a + b_1X_1 + b_2X_2 + b_3X_3 + \cdots + b_nX_n$$

Setiap koefisien menunjukkan kontribusi masing-masing fitur terhadap Y. Pendekatan ini lebih cocok untuk skenario bisnis kompleks dengan banyak faktor penentu.

c) Asumsi Dasar dalam Regresi Linear

Agar hasil prediksi regresi linear dapat dipercaya dan valid, terdapat beberapa asumsi dasar statistik yang harus dipenuhi:

1) Linearitas – hubungan X dan Y bersifat linear.
2) Normalitas residual – selisih antara nilai aktual dan prediksi harus mengikuti distribusi normal.
3) Homoskedastisitas – varians residual harus konstan di seluruh rentang X. Memahami jenis dan asumsi ini penting sebelum mempelajari model machine learning yang lebih kompleks.

Aplikasi Regresi Linear dalam Bisnis Regresi linear bukan hanya konsep statistik, tetapi juga alat penting dalam pengambilan keputusan berbasis data. Beberapa aplikasi umum di dunia bisnis digital meliputi:

2. Aplikasi Regresi Linear dalam Bisnis

Aplikasi Regresi Linear dalam Bisnis Regresi linear bukan hanya konsep statistik, tetapi juga alat penting dalam pengambilan keputusan berbasis data. Beberapa aplikasi umum di dunia bisnis digital meliputi:

a) Prediksi Penjualan Dengan data historis penjualan, perusahaan dapat memprediksi performa masa depan. Model sederhana menggunakan waktu sebagai variabel, sementara model berganda mempertimbangkan faktor tambahan seperti iklan atau diskon untuk proyeksi lebih akurat.
b) Analisis Tren Harga Regresi digunakan untuk memahami arah perubahan harga suatu produk dari waktu ke waktu. Perusahaan dapat menganalisis harga pasar untuk menyusun strategi penetapan harga yang lebih tepat.
c) Peramalan Trafik & Engagement Model regresi dapat digunakan untuk memprediksi trafik situs atau keterlibatan pengguna. Misalnya, startup memanfaatkan data mingguan untuk mempersiapkan kapasitas server atau menyusun strategi peningkatan interaksi berdasarkan faktor-faktor seperti frekuensi konten dan durasi kunjungan.

Regresi linear membantu mengubah data masa lalu menjadi prediksi masa depan yang relevan bagi bisnis, dan sering menjadi dasar untuk pendekatan analitik lanjutan.

3. Implementasi Regresi Linear dengan Scikit-Learn

Implementasi Regresi Linear dengan Scikit-Learn Untuk menerapkan regresi linear secara praktis, Python menyediakan library Scikit-Learn. Library ini memfasilitasi proses pemodelan, mulai dari membagi data, melatih model, membuat prediksi, hingga mengevaluasi performa. Workflow umum terdiri dari lima tahapan yang akan dijelaskan lebih lanjut dalam implementasi teknis.

a) Import Data

Langkah pertama adalah memuat dataset ke dalam notebook. Dataset bisa berupa file .csv dari Google Drive atau data dummy yang dibuat langsung di notebook. Pada tahap ini, data biasanya dimuat ke dalam DataFrame Pandas agar mudah dikelola.

Contoh:

```
import pandas as pd

# Membaca dataset dari file CSV
df = pd.read_csv('/content/drive/MyDrive/dataset_penjualan.csv')

# Melihat 5 baris pertama
print(df.head())
```

Dataset idealnya memiliki kolom fitur (X) seperti jumlah iklan, harga, promosi, dan kolom target (y) seperti jumlah penjualan.

b) Memisahkan Fitur (X) dan Target (y)

Setelah dataset dimuat, langkah berikutnya adalah memisahkan variabel bebas (fitur) dan variabel terikat (target). Target adalah nilai yang ingin diprediksi, sedangkan fitur adalah variabel yang memengaruhi nilai target.

Contoh:

```
# Misalnya kolom 'penjualan' adalah target
X = df[['iklan', 'harga', 'promosi']]
y = df['penjualan']
```

c) Melatih Model (Training) dengan fit()

Berikutnya, data fitur (X) dan target (y) digunakan untuk melatih model regresi linear. Model akan mempelajari hubungan matematis antara X dan y.

Contoh:

```
from sklearn.linear_model import LinearRegression
from sklearn.model_selection import train_test_split

# Membagi data ke data latih (80%) dan uji (20%)
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2,
random_state=42)

# Membuat objek model dan melatihnya
model = LinearRegression()
model.fit(X_train, y_train)
```

Proses fit() ini menghasilkan persamaan regresi (koefisien dan intercept) yang disimpan di dalam objek model.

d) Melakukan Prediksi dengan predict()

Setelah model dilatih, tahap berikutnya adalah menggunakan model untuk memprediksi nilai target pada data uji.

Contoh:

```
# Membuat prediksi pada data uji
y_pred = model.predict(X_test)

# Menampilkan beberapa hasil prediksi vs nilai aktual
for actual, pred in list(zip(y_test[:5], y_pred[:5])):
    print("Aktual:", actual, "| Prediksi:", round(pred, 2))
```

Langkah ini membantu kita mengetahui seberapa baik model bekerja dalam memprediksi data baru yang belum pernah dilihat sebelumnya.

e) Evaluasi Model (R^2, MAE, MSE, RMSE)

Tahap terakhir adalah mengevaluasi performa model menggunakan metrik statistik. Beberapa metrik umum untuk regresi:

- ✓ R^2 (Coefficient of Determination) → Seberapa baik variasi data target dijelaskan oleh model. Semakin mendekati 1 semakin baik.
- ✓ MAE (Mean Absolute Error) → Rata-rata selisih absolut antara nilai aktual dan prediksi.
- ✓ MSE (Mean Squared Error) → Rata-rata kuadrat selisih antara nilai aktual dan prediksi (lebih peka terhadap outlier).
- ✓ RMSE (Root Mean Squared Error) → Akar kuadrat dari MSE, satuannya sama dengan target.

Contoh:

```
from sklearn.metrics import r2_score, mean_absolute_error,
mean_squared_error
import numpy as np

r2 = r2_score(y_test, y_pred)
mae = mean_absolute_error(y_test, y_pred)
mse = mean_squared_error(y_test, y_pred)
rmse = np.sqrt(mse)

print("R²:", round(r2, 3))
print("MAE:", round(mae, 2))
print("MSE:", round(mse, 2))
print("RMSE:", round(rmse, 2))
```

Hasil evaluasi digunakan untuk menilai apakah model sudah cukup akurat untuk mendukung keputusan bisnis. Jika akurasinya rendah, mahasiswa dapat menambahkan fitur, menormalisasi data, atau mencoba algoritma lain. Pola ini merupakan kerangka umum dalam pengembangan model machine learning dan akan mempermudah mahasiswa saat mempelajari model yang lebih kompleks di bab berikutnya.

C. PRAKTIKUM

Pada praktikum ini, mahasiswa akan membangun model regresi linear untuk prediksi data bisnis menggunakan Python di Google Colab. Kegiatan dimulai dengan membuat dataset dummy penjualan, dilanjutkan dengan regresi linear sederhana dan berganda menggunakan Scikit-Learn.

Mahasiswa juga akan menghitung metrik evaluasi seperti R^2, MAE, MSE, dan RMSE, serta memvisualisasikan hasil prediksi. Hasil akhir berupa notebook interaktif yang berisi kode, grafik, dan interpretasi sederhana yang menjadi dasar untuk praktikum lanjutan dalam membangun model prediksi bisnis yang lebih kompleks.

1. Tujuan Praktikum

- ✓ Menerapkan regresi linear sederhana dan berganda menggunakan Scikit-Learn.
- ✓ Menghitung dan menafsirkan hasil evaluasi model (R^2, MAE, MSE, RMSE).
- ✓ Menyajikan hasil prediksi dalam bentuk visualisasi.

2. Studi Kasus

Prediksi Penjualan Produk Berdasarkan Faktor Pemasaran**,** Dataset berisi data dummy penjualan produk selama 12 bulan, dengan kolom:

- bulan → bulan ke-1 sampai ke-12
- iklan → anggaran iklan (juta rupiah)
- harga → harga produk (ribu rupiah)
- penjualan → jumlah unit terjual

3. Langkah-Langkah Praktikum

Membuat Notebook Baru & Import Library

- ➢ Buka https://colab.research.google.com
- ➢ Klik **File** → **New notebook**, beri nama: PraktikumML-Bab4_NIM_NamaLengkap.ipynb
- ➢ Jalankan sel berikut untuk menyiapkan library:

```python
# Versi Python (opsional)
!python --version
# Import library utama
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt

from sklearn.model_selection import train_test_split
from sklearn.linear_model import LinearRegression
from sklearn.metrics import r2_score, mean_absolute_error, mean_squared_error

# Pengaturan tampilan pandas (opsional)
pd.set_option('display.max_columns', None)
pd.set_option('display.precision', 3)
```

```
Python 3.12.11
```

Membuat Dataset Dummy (Contoh Tetap, 12 Baris)

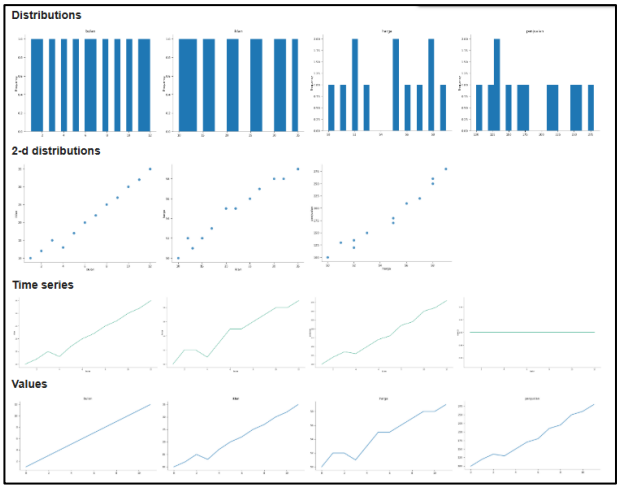
Keterangan Kolom:

- bulan → indikator waktu (opsional)
- iklan → biaya iklan per bulan (juta rupiah) — **fitur**
- harga → harga produk (ribu rupiah) — **fitur**
- penjualan → unit terjual — **target**

```python
# Dataset dummy penjualan (12 bulan)
data = {
    "bulan": np.arange(1, 13),
    "iklan": [10, 12, 15, 13, 17, 20, 22, 25, 27, 30, 32, 35],
    "harga": [50, 52, 52, 51, 53, 55, 55, 56, 57, 58, 58, 59],
    "penjualan": [100, 120, 135, 130, 150, 170, 180, 210, 220, 250, 260, 280]
}
df = pd.DataFrame(data)
df
```

Ketika di ekseskusi dan di jalankan kode program tersebut akan menghasilkan output seperti berikut ini

	bulan	iklan	harga	penjualan
0	1	10	50	100
1	2	12	52	120
2	3	15	52	135
3	4	13	51	130
4	5	17	53	150
5	6	20	55	170
6	7	22	55	180
7	8	25	56	210
8	9	27	57	220
9	10	30	58	250
10	11	32	58	260
11	12	35	59	280



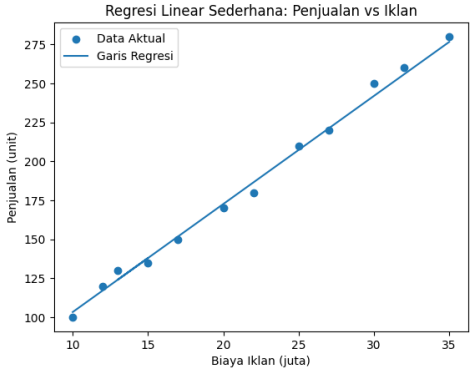
Regresi Linear Sederhana

Memprediksi penjualan berdasarkan iklan. (X = iklan, y = penjualan) Langkah: pisahkan fitur & target → split train/test → latih model → prediksi → evaluasi → visualisasi.

```python
# Pilih dataset yang akan digunakan (pilih salah satu):
# df_used = df_rand       # gunakan untuk dataset acak
df_used = df

# Fitur & target
X = df_used[['iklan']]   # fitur tunggal
y = df_used['penjualan'] # target

# Split data
X_train, X_test, y_train, y_test = train_test_split(
    X, y, test_size=0.2, random_state=42
)

# Model
model_sederhana = LinearRegression()
model_sederhana.fit(X_train, y_train)

# Prediksi
y_pred = model_sederhana.predict(X_test)

# Evaluasi
r2 = r2_score(y_test, y_pred)
mae = mean_absolute_error(y_test, y_pred)
mse = mean_squared_error(y_test, y_pred)
rmse = np.sqrt(mse)

print("=== Evaluasi: Regresi Linear Sederhana ===")
print("R²   :", round(r2, 3))
print("MAE  :", round(mae, 3))
print("MSE  :", round(mse, 3))
print("RMSE :", round(rmse, 3))

# Koefisien & intercept
print("\nKoefisien (slope):", model_sederhana.coef_[0])
print("Intercept (a):", model_sederhana.intercept_)

# Visualisasi garis regresi
plt.scatter(X, y, label='Data Aktual')
plt.plot(X, model_sederhana.predict(X), label='Garis Regresi')
plt.xlabel("Biaya Iklan (juta)")
plt.ylabel("Penjualan (unit)")
plt.title("Regresi Linear Sederhana: Penjualan vs Iklan")
plt.legend()
plt.show()
```

Kode program tersebut akan menghasilkan output seperti berikut ini

```
=== Evaluasi: Regresi Linear Sederhana ===
R²   : 0.994
MAE  : 5.192
MSE  : 31.269
RMSE : 5.592

Koefisien (slope): 6.931988041853512
Intercept (a): 33.96113602391631
```



Regresi Linear Berganda (X = iklan, harga → y = penjualan)

Tambahkan lebih dari satu fitur untuk memodelkan pengaruh ganda terhadap target.

```
# Gunakan dataset tetap 12 baris (df) untuk contoh ini
X_multi = df[['iklan', 'harga']]
y_multi = df['penjualan']

# Split
X_train_m, X_test_m, y_train_m, y_test_m = train_test_split(
    X_multi, y_multi, test_size=0.2, random_state=42
)

# Model
model_ganda = LinearRegression()
model_ganda.fit(X_train_m, y_train_m)

# Prediksi
y_pred_m = model_ganda.predict(X_test_m)

# Evaluasi
r2_m = r2_score(y_test_m, y_pred_m)
mae_m = mean_absolute_error(y_test_m, y_pred_m)
mse_m = mean_squared_error(y_test_m, y_pred_m)
rmse_m = np.sqrt(mse_m)

print("=== Evaluasi: Regresi Linear Berganda ===")
print("R²   :", round(r2_m, 3))
print("MAE  :", round(mae_m, 3))
print("MSE  :", round(mse_m, 3))
print("RMSE :", round(rmse_m, 3))

# Koefisien & intercept
coef_series = pd.Series(model_ganda.coef_, index=X_multi.columns, name='Koefisien')
display(coef_series.to_frame())
print("Intercept (a):", model_ganda.intercept_)

# Tabel perbandingan aktual vs prediksi
hasil = pd.DataFrame({"Aktual": y_test_m.values, "Prediksi": np.round(y_pred_m, 2)})
hasil.reset_index(drop=True, inplace=True)
hasil
```

Ketika di ekseskusi dan di jalankan kode program tersebut akan menghasilkan output seperti berikut ini

```
=== Evaluasi: Regresi Linear Berganda ===
R²   : 0.993
MAE  : 5.763
MSE  : 38.81
RMSE : 6.23
```

	Koefisien
iklan	7.860
harga	-2.665

```
Intercept (a): 159.89671361502369
```

	Aktual	Prediksi
0	260	256.83
1	250	241.11
2	100	105.23

Menyimpan Hasil Prediksi ke CSV

Colab menjalankan file .ipynb di lingkungan runtime sementara (virtual machine cloud). File yang disimpan tanpa path lengkap otomatis berada di /content/, dan tidak muncul di Google Drive kecuali Anda memindahkannya secara eksplisit.

```
# Simpan hasil regresi sederhana
hasil_sederhana = pd.DataFrame({
    "Aktual": y_test.values,
    "Prediksi": np.round(y_pred, 2)
})
hasil_sederhana.to_csv("hasil_regresi_sederhana.csv", index=False)

# Simpan hasil regresi berganda
hasil_berganda = pd.DataFrame({
    "Aktual": y_test_m.values,
    "Prediksi": np.round(y_pred_m, 2)
})
hasil_berganda.to_csv("hasil_regresi_berganda.csv", index=False)
print("File tersimpan: hasil_regresi_sederhana.csv, hasil_regresi_berganda.csv")
```

```
File tersimpan: hasil_regresi_sederhana.csv, hasil_regresi_berganda.csv
```

```
!ls -l /content/*.csv # Melihat File yang Tersimpan di /content/
```

```
-rw-r--r-- 1 root root 49 Sep 22 11:49 /content/hasil_regresi_berganda.csv
-rw-r--r-- 1 root root 49 Sep 22 11:49 /content/hasil_regresi_sederhana.csv
```

Menyimpan & Membagikan Notebook

a) Klik **File → Save a copy in Drive**.
b) Klik **Share → Anyone with the link → Viewer**.
c) Kirim tautan notebook ke LMS atau ke dosen sesuai petunjuk pengumpulan.

D. LATIHAN DAN TUGAS

1. Latihan Regresi Linear Sederhana

Tujuan: Mahasiswa mampu membangun model regresi linear sederhana untuk memprediksi penjualan berdasarkan satu variabel.

Instruksi:

a) Buat notebook baru di Colab:
b) ML-Bab4-Latihan_NIM_NamaLengkap.ipynb
c) Buat dataset dummy minimal 15 baris dengan kolom:
   - biaya_iklan (juta rupiah)
   - penjualan (unit produk terjual)
d) Lakukan tahapan:
   - Import library (pandas, sklearn, matplotlib).
   - Pisahkan fitur (X) dan target (y).
   - Bagi data (train_test_split).
   - Latih model regresi linear (LinearRegression + .fit()).
   - Lakukan prediksi (.predict()).
   - Hitung metrik evaluasi: R^2, MAE, MSE, RMSE.
   - Visualisasikan grafik scatter data aktual dan garis regresi.
e) Simpan di Google Drive dan **bagikan (Viewer)** ke dosen.

2. Tugas

Manfaat Regresi Linear dalam Bisnis Digital

Kerjakan secara individu. Jawab dalam 1 file dokumen (PDF atau langsung di Google Docs). Setiap jawaban minimal 1 paragraf (5–7 kalimat) dan gunakan bahasa ilmiah yang baik. Sertakan identitas di bagian atas (Nama, NIM, Prodi, Kelas). Upload di LMS sesuai petunjuk dosen.

Jawablah pertanyaan berikut secara naratif:

1) Jelaskan dengan bahasa Anda sendiri apa itu regresi linear, serta perbedaan antara regresi linear sederhana dan regresi linear berganda, lengkap dengan contoh kasus bisnis digital.
2) Sebutkan dan jelaskan tiga asumsi dasar yang harus dipenuhi dalam membangun model regresi linear. Mengapa pelanggaran terhadap asumsi ini dapat memengaruhi keakuratan model?
3) Jelaskan fungsi dan makna dari metrik R^2, MAE, MSE, dan RMSE. Sertakan contoh interpretasi sederhana untuk setiap metrik jika hasilnya tinggi atau rendah.
4) Uraikan bagaimana regresi linear dapat membantu perusahaan digital dalam pengambilan keputusan berbasis data. Sertakan minimal 2 contoh penerapan nyata.
5) Anda memiliki data historis penjualan 12 bulan terakhir. Jelaskan langkah-langkah yang akan Anda lakukan untuk membangun model regresi linear sederhana menggunakan Python & Scikit-Learn.
6) Setelah membangun model, ternyata nilai R^2 sangat rendah. Apa kemungkinan penyebabnya dan langkah perbaikan apa yang bisa dilakukan?
7) Menurut Anda, apa kelebihan dan keterbatasan model regresi linear jika digunakan dalam skenario bisnis digital yang kompleks?
8) Bagaimana regresi linear dapat membantu pengambilan keputusan bisnis?
9) Apa keterbatasannya jika diterapkan di industri nyata?
10) Kapan kita harus mulai mempertimbangkan model lain (non-linear)?

MODUL V
KLASIFIKASI: MENGENAL POLA DALAM DATA

A. CAPAIAN PEMBELAJARAN

1) Mahasiswa memahami konsep dasar klasifikasi dalam machine learning.
2) Mahasiswa mengenal algoritma K-Nearest Neighbor (KNN), Decision Tree, dan Random Forest.
3) Mahasiswa mampu menerapkan algoritma klasifikasi untuk kasus bisnis digital.
4) Mahasiswa dapat mengevaluasi hasil klasifikasi menggunakan metrik evaluasi.

B. DASAR TEORI

1. Konsep Klasifikasi

Klasifikasi adalah metode supervised learning untuk memprediksi kelas dari data baru berdasarkan pola dari data berlabel. Berbeda dari regresi yang memprediksi nilai numerik, klasifikasi menghasilkan kategori diskrit (misalnya: spam/tidak spam, churn/loyal). Prosesnya meliputi pelatihan model dari data berlabel, memprediksi kelas data baru, dan mengevaluasi hasilnya. Terdapat dua tipe klasifikasi:

a) Binary Classification (dua kelas)
b) Multiclass Classification (lebih dari dua kelas)

Contoh Penerapan:

- ✓ Email: spam vs non-spam
- ✓ Pelanggan: churn vs loyal
- ✓ Medis: positif vs negatif COVID-19

Klasifikasi sangat penting dalam bisnis karena dapat membantu: Mendeteksi pelanggan yang berisiko churn, Mengidentifikasi fraud, Mengelompokkan umpan balik pelanggan, Mendukung pengambilan keputusan berbasis data

2. Algoritma Klasifikasi Populer

Beberapa algoritma klasifikasi umum dan efektif yang sering digunakan antara lain:

a) K-Nearest Neighbor (KNN): KNN mengklasifikasi data baru berdasarkan kedekatan dengan data berlabel terdekat. Semakin dekat titik data, semakin besar kemungkinan mereka berada di kelas yang sama. Kelebihannya sederhana dan intuitif, tapi kurang efisien untuk dataset besar dan sensitif terhadap skala fitur.
b) Decision Tree Decision: Tree membagi data berdasarkan fitur secara bertahap hingga menghasilkan keputusan. Setiap node adalah pertanyaan, dan daun pohon adalah output klasifikasi. Mudah dipahami dan divisualisasikan, namun rentan overfitting jika tidak dikendalikan.
c) Random Forest: Random Forest menggabungkan banyak decision tree untuk meningkatkan akurasi dan stabilitas prediksi. Lebih tahan terhadap overfitting dibanding satu pohon, tetapi sulit diinterpretasikan dan membutuhkan komputasi lebih besar.

3. Contoh Kasus Nyata Klasifikasi

a) Prediksi Churn Pelanggan Mengklasifikasi pelanggan menjadi churn atau loyal berdasarkan data transaksi, durasi langganan, dan respons terhadap promosi untuk menyusun strategi retensi.

b) Deteksi Penipuan Membedakan transaksi normal dan mencurigakan berdasarkan pola perilaku pengguna. Model ini penting untuk keamanan finansial dan biasanya berjalan secara real-time.
c) Klasifikasi Dokumen Mengelompokkan dokumen ke dalam kategori seperti keuangan, legal, atau pemasaran berdasarkan isi teks. Digunakan di perusahaan, media, dan lembaga pemerintah.
d) Diagnosis Kesehatan Memungkinkan identifikasi dini penyakit berdasarkan data pasien. Aplikasi ini membantu dalam pengambilan keputusan medis, meskipun memerlukan validasi ketat.

Contoh klasifikasi churn pelanggan akan digunakan dalam praktikum karena relevan dan mudah dipahami.

4. Evaluasi Model Klasifikasi

Evaluasi Model Klasifikasi Evaluasi penting untuk mengukur kinerja model klasifikasi. Berbeda dari regresi, klasifikasi dievaluasi menggunakan metrik berbasis jumlah prediksi benar dan salah. Beberapa metrik utama:

a) Confusion Matrix

Confusion matrix merupakan representasi visual berupa tabel yang menunjukkan jumlah prediksi benar dan salah dari sebuah model klasifikasi. Matriks ini sangat berguna untuk memahami jenis kesalahan apa yang dilakukan model, bukan hanya berapa banyak kesalahan yang terjadi. Untuk kasus binary classification (dua kelas), confusion matrix terdiri atas empat komponen:

	Prediksi Positif	**Prediksi Negatif**
Aktual Positif	True Positive (TP)	False Negative (FN)
Aktual Negatif	False Positive (FP)	True Negative (TN)

- True Positive (TP) → model memprediksi positif dan benar-benar positif
- True Negative (TN) → model memprediksi negatif dan benar-benar negatif
- False Positive (FP) → model memprediksi positif padahal sebenarnya negatif (kesalahan tipe I)
- False Negative (FN) → model memprediksi negatif padahal sebenarnya positif (kesalahan tipe II)

Dari keempat nilai ini, kita dapat menghitung berbagai metrik evaluasi lainnya.

b) Akurasi, Precision, Recall, dan F1-Score

Empat metrik ini adalah yang paling sering digunakan dalam mengevaluasi model klasifikasi:

- **Akurasi** (Accuracy)
  Mengukur proporsi prediksi yang benar dibandingkan total data.

$$\textbf{Akurasi} = \frac{TP + TN}{TP + TN + FP + FN}$$

Akurasi cocok digunakan saat jumlah data di setiap kelas seimbang, namun bisa menyesatkan saat data imbalance (misalnya kasus penipuan 1% vs transaksi normal 99%).

- **Precision** (Presisi)
  Mengukur seberapa tepat model dalam memprediksi kelas positif.

$$\textbf{Precision} = \frac{TP}{TP + FP}$$

Precision tinggi berarti sedikit kesalahan positif palsu. Penting pada kasus seperti deteksi spam, karena tidak ingin terlalu banyak email normal yang dikira spam.

- **Recall** (Sensitivitas)
  Mengukur seberapa banyak data positif yang berhasil ditemukan model.

$$\textbf{Recall} = \frac{TP}{TP + FN}$$

Recall tinggi berarti sedikit positif yang terlewat. Penting pada kasus deteksi penyakit, karena tidak boleh ada pasien sakit yang terlewatkan.

- **F1-Score**
  Merupakan rata-rata harmonis dari precision dan recall, digunakan untuk mencari keseimbangan keduanya:

$$\mathbf{F1} = 2 \times \frac{Precision \times Recall}{Precision + Recall}$$

Nilai F1 tinggi menunjukkan model mampu menemukan data positif dengan tepat sekaligus tidak banyak melakukan kesalahan.

c) ROC Curve dan AUC (Pengantar Singkat)

Selain confusion matrix dan metrik turunannya, ada satu metode visual populer lain untuk mengevaluasi model klasifikasi, yaitu ROC Curve (Receiver Operating Characteristic). Kurva ROC menampilkan hubungan antara True Positive Rate (Recall) dan False Positive Rate pada berbagai ambang batas (threshold) probabilitas prediksi. Luas area di bawah kurva ROC disebut AUC (Area Under the Curve). Nilai AUC berada antara 0 hingga 1:

- AUC = 0.5 berarti performa model setara dengan tebak-tebakan acak
- AUC mendekati 1 menunjukkan model sangat baik dalam membedakan kelas positif dan negatif

ROC-AUC sangat berguna saat kita ingin membandingkan beberapa model klasifikasi secara keseluruhan, terutama ketika data bersifat imbalanced dan akurasi saja tidak cukup representatif. Memahami berbagai metrik ini akan membantu mahasiswa tidak hanya membangun model klasifikasi, tetapi juga mampu menilai kualitas dan reliabilitas model tersebut secara objektif. Pada bagian praktikum nanti, setelah membangun model KNN, Decision Tree, dan Random Forest, kita akan mengevaluasi performa ketiganya menggunakan confusion matrix dan metrik evaluasi ini untuk menentukan model mana yang paling efektif.

C. PRAKTIKUM

Pada bagian praktikum ini, mahasiswa akan membangun model klasifikasi sederhana untuk memprediksi status pelanggan: loyal atau churn, menggunakan tiga algoritma populer yaitu K-Nearest Neighbor (KNN), Decision Tree, dan Random Forest.

Tujuan dari praktikum ini adalah agar mahasiswa dapat memahami alur kerja dasar machine learning untuk klasifikasi, mulai dari membuat dataset, melatih model, memprediksi data baru, hingga mengevaluasi performa model menggunakan confusion matrix dan metrik evaluasi.

1. Persiapan Dataset: Data Pelanggan (Churn vs Loyal)

Sebelum membangun model, kita perlu menyiapkan dataset. Dataset ini akan dibuat secara manual (dummy) menggunakan Python agar mahasiswa memahami struktur data yang dibutuhkan.

- Buka https://colab.research.google.com
- Klik **File** → **New notebook**, beri nama: PraktikumML-Bab5_NIM_NamaLengkap.ipynb
- Buat script python seperti berikut dan jalankan untuk membuat Dataset Data Pelanggan (Churn vs Loyal)

```
# --- Membuat Dataset Dummy Pelanggan (Churn vs Loyal) ---
import pandas as pd
import numpy as np
from google.colab import files

# Ganti angka seed berikut sesuai kebutuhan:
# - Praktikum kelas (seragam): gunakan 42
# - Tugas individu: ganti dengan nomor absen Anda (misal 7, 23, dst.)
STUDENT_SEED = 42
np.random.seed(STUDENT_SEED)

# Membuat 30 data pelanggan dummy
data = {
    'Lama_Berlangganan_bulan': np.random.randint(1, 36, 30),
    'Frekuensi_Pembelian': np.random.randint(1, 20, 30),
    'Total_Pengeluaran_juta': np.round(np.random.uniform(1, 50, 30), 2),
    'Respon_Promo': np.random.randint(0, 2, 30),  # 0 = tidak responsif, 1 = responsif
    'Status': np.random.choice(['Loyal', 'Churn'], 30)
}

df = pd.DataFrame(data)
df.to_csv('data_pelanggan.csv', index=False)

print('Contoh 5 baris pertama dataset:')
display(df.head())

# --- Unduh langsung (opsional) ---
!ls -l /content/*.csv  # melihat file CSV di /content/
print('\n Mengunduh file ke perangkat lokal...')
files.download('data_pelanggan.csv')
print(' File berhasil dikirim untuk diunduh.')
```

Catatan: Dataset ini memiliki 4 fitur utama dan 1 label target (Status). Label memiliki dua kelas: **"Loyal" dan "Churn"**.

```
Contoh 5 baris pertama dataset:
```

	Lama_Berlangganan_bulan	Frekuensi_Pembelian	Total_Pengeluaran_juta	Respon_Promo	Status
0	29	4	28.95	1	Loyal
1	15	14	26.52	1	Churn
2	8	18	48.10	1	Loyal
3	21	9	42.38	0	Churn
4	19	2	37.62	0	Churn

```
-rw-r--r-- 1 root root 657 Sep 13 14:25 /content/data_pelanggan.csv

 Mengunduh file ke perangkat lokal...
 File berhasil dikirim untuk diunduh.
```

Penjelasan Struktur Dataset

- Fitur(X): Lama_Berlangganan_bulan, Frekuensi_Pembelian, Total_Pengeluaran_juta , Respon_Promo
- Target(y): Status ∈ {Loyal, Churn}

- Untuk KNN, skala fitur berpengaruh, maka kita standarisasi fitur numerik.

Berikut adalah dataset hasil generate yang di buat menggunakan script python di atas

Lama Berlangganan (bulan)	Frekuensi Pembelian	Total Pengeluaran (juta)	Respon Promo	Status
29	4	28.95	1	Loyal
15	14	26.52	1	Churn
8	18	48.1	1	Loyal
21	9	42.38	0	Churn
19	2	37.62	0	Churn
23	15	27.44	0	Churn
11	7	29.75	0	Loyal
11	12	48.3	0	Loyal
24	8	30.74	0	Loyal
24	15	14.52	1	Loyal
3	3	15.52	0	Loyal
22	14	9.1	0	Loyal
2	17	1.77	0	Loyal
24	4	21.75	1	Churn
30	18	20.35	0	Loyal
2	8	15.38	0	Churn
21	4	1.69	1	Loyal
33	2	10.74	0	Loyal
12	6	35.86	0	Loyal
22	10	39.72	0	Churn
25	4	30.69	0	Churn
27	18	46.39	0	Churn
28	12	32.9	1	Churn
16	2	45.83	1	Loyal
15	10	42.65	1	Churn
3	4	23.02	0	Loyal
7	14	5.68	0	Loyal
21	16	19.17	0	Churn
9	15	33.77	1	Churn
18	8	33.63	0	Churn

2. Train/Test Split & Standarisasi

Sebelum membangun model klasifikasi, dataset perlu dibagi menjadi data latih (train) dan data uji (test).

- Data latih digunakan untuk melatih model agar mengenali pola dari data
- Data uji digunakan untuk mengetes kemampuan model memprediksi data baru

Dengan pembagian ini, kita dapat menilai apakah model benar-benar mampu melakukan generalisasi, bukan sekadar menghafal data latih. Selain itu, karena algoritma seperti KNN sangat sensitif terhadap skala nilai fitur, kita juga perlu melakukan standarisasi (standardization) — yaitu menyamakan skala seluruh fitur numerik agar tidak ada yang mendominasi perhitungan jarak.

Standarisasi dilakukan hanya pada data latih (fit & transform), lalu dilanjutkan ke data uji (transform saja) agar proses evaluasi tetap adil.

```
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler

# Pisahkan fitur dan target
X = df.drop('Status', axis=1)
y = df['Status']

# Bagi data: 80% train, 20% test
X_train, X_test, y_train, y_test = train_test_split(
    X, y, test_size=0.2, random_state=42, stratify=y
)

# Standarisasi untuk KNN (opsional untuk tree-based)
scaler = StandardScaler()
X_train_s = scaler.fit_transform(X_train)
X_test_s  = scaler.transform(X_test)

prin('Ukuran data latih:', X_train.shape, '| data uji:', X_test.shape)
```

```
Ukuran data latih: (24, 4) | data uji: (6, 4)
```

Artinya:

- **Data latih (train set)** terdiri dari **24 baris** (sampel) dan **4 kolom** (fitur).
- **Data uji (test set)** terdiri dari **6 baris** (sampel) dan **4 kolom** (fitur).

Kenapa jadi 24 dan 6? Karena dataset awal punya **30 baris data**. Dengan test_size=0.2 → 20% untuk uji = 6 baris, sisanya 80% untuk latih = 24 baris. Empat kolom (fitur) itu adalah variabel input (Lama_Berlangganan_bulan, Frekuensi_Pembelian, Total_Pengeluaran_juta, Respon_Promo), sedangkan kolom Status (Loyal/Churn) dipisahkan sebagai **label/target** (y_train dan y_test).

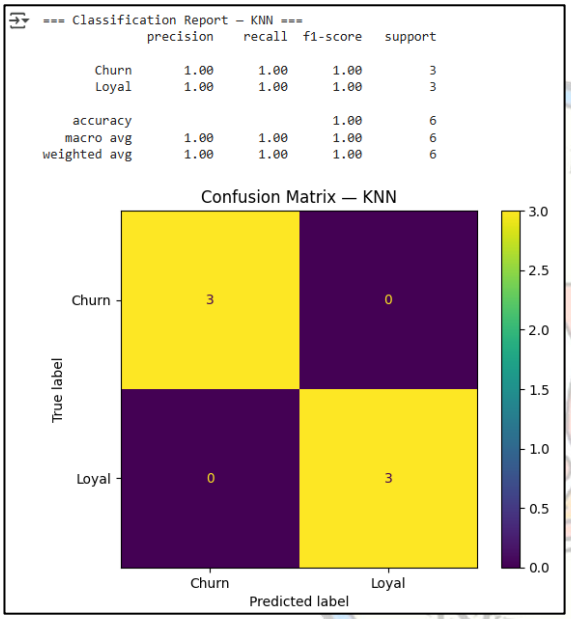
3. Praktikum 1 — K-Nearest Neighbor (KNN)

Pada praktikum ini, mahasiswa akan mempelajari cara kerja algoritma KNN untuk klasifikasi data sederhana. KNN mengklasifikasikan data baru berdasarkan mayoritas kelas dari k tetangga terdekatnya. Model akan dilatih menggunakan data pelanggan yang telah distandarisasi, lalu diuji pada data baru.Evaluasi dilakukan menggunakan classification report (precision, recall, f1-score, accuracy) dan confusion matrix untuk melihat performa model dalam membedakan kelas.

```
# Import library yang dibutuhkan
from sklearn.neighbors import KNeighborsClassifier   # Algoritma KNN
# Metrik evaluasi
from sklearn.metrics import accuracy_score, classification_report, confusion_matrix, ConfusionMatrixDisplay
import matplotlib.pyplot as plt                      # Untuk visualisasi grafik

# Inisialisasi model KNN dengan k = 5 (jumlah tetangga terdekat yang dipakai)
knn = KNeighborsClassifier(n_neighbors=5)

# Latih model KNN menggunakan data latih yang sudah distandarisasi
# X_train_s = fitur (data latih), y_train = label (target kelas)
knn.fit(X_train_s, y_train)

# Gunakan model yang sudah dilatih untuk memprediksi kelas data uji
# X_test_s = fitur (data uji yang sudah distandarisasi)
y_pred_knn = knn.predict(X_test_s)
```

```python
# Cetak laporan klasifikasi (precision, recall, f1-score, accuracy)
print('=== Classification Report — KNN ===')
print(classification_report(y_test, y_pred_knn))

# Hitung confusion matrix → tabel perbandingan label aktual vs hasil prediksi
cm_knn = confusion_matrix(y_test, y_pred_knn, labels=knn.classes_)

# Buat objek visualisasi confusion matrix dengan label kelas
disp = ConfusionMatrixDisplay(confusion_matrix=cm_knn, display_labels=knn.classes_)

# Tampilkan confusion matrix dalam bentuk grafik (warna)
disp.plot()
plt.title('Confusion Matrix — KNN')  # Judul grafik
plt.show()                           # Render grafik ke layar
```

Setelah di jalanakn scrip tersebut akan menghasilkan output sebagai berikut:



Classification Report

Bagian ini menampilkan **metrik evaluasi utama** untuk model klasifikasi:

- **Precision** → dari semua prediksi "positif" untuk suatu kelas, berapa yang benar? Misal: dari semua prediksi "Churn", ternyata semuanya benar (precision = 1.00).
- **Recall** → dari semua data yang sebenarnya termasuk suatu kelas, berapa yang berhasil ditebak dengan benar oleh model?
  Misal: semua data "Churn" terprediksi benar (recall = 1.00).
- **F1-score** → gabungan antara precision dan recall, nilai terbaik = 1.00.
  Pada hasil ini, baik kelas **Churn** maupun **Loyal** mendapat **1.00 penuh**.
- **Support** → jumlah data aktual di tiap kelas pada data uji.
  Terlihat ada **3 data Churn** dan **3 data Loyal** (total 6 data uji).

Karena semua metrik = 1.00 (100%), artinya model KNN memprediksi semua data uji dengan sempurna. **Confusion Matrix** Tabel 2×2 ini memperlihatkan detail hasil prediksi:

	Prediksi Churn	Prediksi Loyal
Aktual Churn	3	0
Aktual Loyal	0	3

Interpretasi:

- 3 data Churn diprediksi benar sebagai Churn
- 3 data Loyal diprediksi benar sebagai Loyal

- Tidak ada kesalahan prediksi (0 error)

Warna kuning di diagonal menunjukkan jumlah prediksi benar, warna ungu gelap menunjukkan 0 kesalahan.

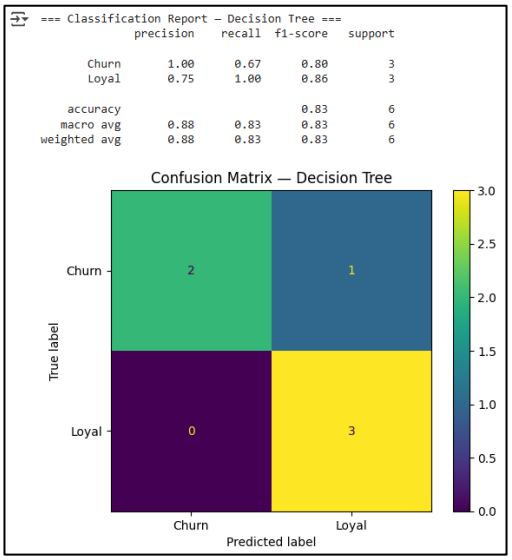
4. Praktikum 2 — Decision Tree

Pada praktikum ini, mahasiswa akan menggunakan algoritma Decision Tree untuk klasifikasi data pelanggan. Model dilatih dari data latih dan diuji pada data baru, dengan hasil dievaluasi menggunakan classification report dan confusion matrix. Decision Tree membentuk struktur pohon keputusan berdasarkan kondisi fitur, sehingga mudah dipahami dan divisualisasikan seperti aturan if-else.

```
# Import library yang diperlukan
from sklearn.tree import DecisionTreeClassifier     # Algoritma Decision Tree
# mengimpor tiga fungsi/kelas penting dari modul sklearn.metrics
# digunakan untuk mengevaluasi performa model klasifikasi
from sklearn.metrics import classification_report, confusion_matrix, ConfusionMatrixDisplay
import matplotlib.pyplot as plt

# Inisialisasi model Decision Tree
# criterion='gini' → menggunakan indeks Gini untuk mengukur kualitas split
# random_state=42 → memastikan hasil replikasi tetap sama
dtree = DecisionTreeClassifier(criterion='gini', random_state=42)

# Latih model menggunakan data latih (fit)
dtree.fit(X_train, y_train)

# Lakukan prediksi pada data uji
y_pred_dt = dtree.predict(X_test)

# Cetak laporan klasifikasi (precision, recall, f1-score, accuracy)
print("=== Classification Report – Decision Tree ===")
print(classification_report(y_test, y_pred_dt))

# Buat confusion matrix dari hasil prediksi
cm_dt = confusion_matrix(y_test, y_pred_dt, labels=dtree.classes_)

# Visualisasikan confusion matrix
disp = ConfusionMatrixDisplay(confusion_matrix=cm_dt, display_labels=dtree.classes_)
disp.plot()
plt.title("Confusion Matrix – Decision Tree")
plt.show()
```

Setelah di jalanakn scrip tersebut akan menghasilkan output sebagai berikut:

```
=== Classification Report – Decision Tree ===
              precision    recall  f1-score   support

       Churn       1.00      0.67      0.80         3
       Loyal       0.75      1.00      0.86         3

    accuracy                           0.83         6
   macro avg       0.88      0.83      0.83         6
weighted avg       0.88      0.83      0.83         6
```



Interpretasi Hasil Classification Report

Churn

- **Precision = 1.00** → semua prediksi "Churn" benar (tidak ada yang salah ditebak Loyal).
- **Recall = 0.67** → dari 3 data Churn aktual, hanya 2 yang berhasil ditebak benar → 1 salah diprediksi sebagai Loyal.
- **F1-score = 0.80** → kombinasi precision & recall, cukup tinggi tapi tidak sempurna.

Loyal

- **Precision = 0.75** → dari semua prediksi "Loyal", ada sebagian salah (1 data Churn ikut masuk).
- **Recall = 1.00** → semua data Loyal berhasil diprediksi dengan benar (tidak ada yang miss).
- **F1-score = 0.86** → cukup baik.
- **Accuracy = 0.83 (83%)** → dari 6 data uji, ada 5 prediksi benar dan 1 salah.
- **Macro avg / Weighted avg** = 0.83 → rata-rata performa model di kedua kelas.

Confusion Matrix (Tabel hasil prediksi Decision Tree)

	Prediksi Churn	Prediksi Loyal
Aktual Churn	2	1
Aktual Loyal	0	3

- Dari 3 data **Churn aktual**, model menebak 2 dengan benar, 1 salah ke Loyal.
- Dari 3 data **Loyal aktual**, model menebak semuanya dengan benar (3/3).
- Kotak kuning diagonal (2 + 3) = jumlah prediksi benar = 5.
- Kotak biru (1) = salah klasifikasi (false negative untuk Churn).

Kesimpulan

- ✓ Model **Decision Tree** cukup baik dengan akurasi 83%, tetapi masih membuat **satu kesalahan pada kelas Churn**.
- ✓ Model lebih **unggul pada kelas Loyal (recall 100%)**, namun sedikit kurang konsisten dalam mengenali semua Churn.
- ✓ Dalam kasus nyata, ini berarti model **cenderung lebih hati-hati mendeteksi Loyal** dan bisa saja kurang sensitif terhadap beberapa pelanggan yang sebenarnya Churn.

Mengapa hasilnya berbeda?

Meskipun datasetnya sama, hasil bisa berbeda karena **cara kerja algoritmanya berbeda**:

- **KNN (K-Nearest Neighbor):** Menentukan kelas sebuah data berdasarkan mayoritas *tetangga terdekat* (berdasarkan jarak). Performa sangat dipengaruhi oleh **skala data** (itulah kenapa perlu standarisasi). Karena dataset kita kecil dan cukup terpisah jelas, KNN bisa menghasilkan **akurasi sempurna (100%)**.
- **Decision Tree:** Membagi data berdasarkan aturan (*if-else*) menggunakan kriteria tertentu (misalnya Gini Index). Sangat mudah dipahami, tapi rentan **overfitting** atau salah memotong data ketika jumlah data sedikit. Pada dataset ini, satu data Churn salah diklasifikasikan → akurasi turun menjadi **83%**.

Jadi, meskipun datanya sama, algoritma yang berbeda menghasilkan hasil evaluasi yang berbeda.

Mana yang lebih baik? Jawabannya: tergantung kondisi dataset & tujuan analisis. Pada dataset kecil dengan fitur numerik terstandarisasi seperti ini → KNN tampil lebih baik (100%). Pada dataset lebih besar, dengan kombinasi variabel numerik & kategorikal, atau ketika interpretabilitas penting → Decision Tree lebih berguna, karena bisa menjelaskan "aturan"

klasifikasi. Dalam dunia nyata, tidak ada algoritma yang selalu terbaik. Biasanya, peneliti/data scientist akan mencoba beberapa algoritma lalu membandingkan performanya (accuracy, precision, recall, dll.), kemudian memilih yang paling sesuai dengan kebutuhan bisnis.

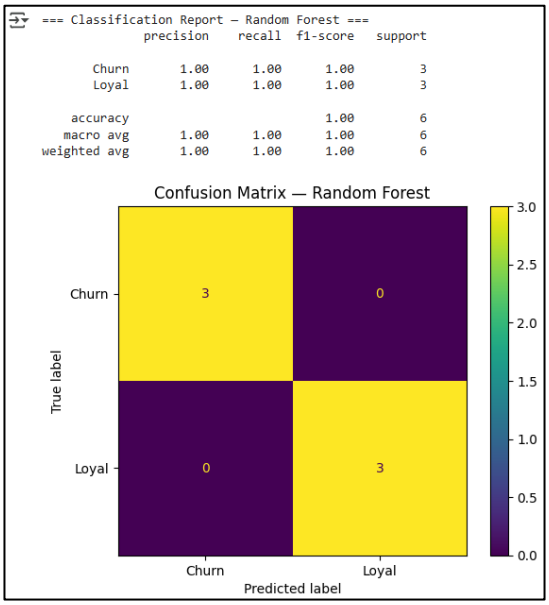
5. Praktikum 3 — Random Forest

Dalam praktikum ini, mahasiswa akan menerapkan algoritma Random Forest untuk klasifikasi data pelanggan. Algoritma ini menggabungkan banyak Decision Tree dan menentukan prediksi akhir lewat voting mayoritas, sehingga lebih akurat dan tahan overfitting. Model akan dilatih, diuji, dievaluasi dengan classification report, dan divisualisasikan melalui confusion matrix.

```
# Import library yang dibutuhkan
from sklearn.ensemble import RandomForestClassifier       # Algoritma Random Forest
from sklearn.metrics import classification_report, confusion_matrix, ConfusionMatrixDisplay
import matplotlib.pyplot as plt

# Inisialisasi model Random Forest
# n_estimators=100 → jumlah pohon keputusan yang digunakan
# random_state=42 → memastikan hasil tetap konsisten
rf = RandomForestClassifier(n_estimators=100, random_state=42)

# Latih model menggunakan data latih (fit)
rf.fit(X_train, y_train)

# Lakukan prediksi pada data uji
y_pred_rf = rf.predict(X_test)

# Cetak laporan klasifikasi (precision, recall, f1-score, accuracy)
print("=== Classification Report – Random Forest ===")
print(classification_report(y_test, y_pred_rf))

# Buat confusion matrix dari hasil prediksi
cm_rf = confusion_matrix(y_test, y_pred_rf, labels=rf.classes_)

# Visualisasikan confusion matrix
disp = ConfusionMatrixDisplay(confusion_matrix=cm_rf, display_labels=rf.classes_)
disp.plot()
plt.title("Confusion Matrix – Random Forest")
plt.show()
```

Setelah di jalanakn scrip tersebut akan menghasilkan output sebagai berikut:

```
=== Classification Report – Random Forest ===
              precision    recall  f1-score   support

       Churn       1.00      1.00      1.00         3
       Loyal       1.00      1.00      1.00         3

    accuracy                           1.00         6
   macro avg       1.00      1.00      1.00         6
weighted avg       1.00      1.00      1.00         6
```



Interpretasi hasil analisis dataset

Churn

- Precision = 1.00 → semua data yang diprediksi "Churn" memang benar-benar Churn (tidak ada salah tebak).
- Recall = 1.00 → dari semua data Churn aktual, semuanya berhasil diprediksi benar.
- F1-score = 1.00 → kombinasi precision & recall sempurna.

Loyal

- Precision = 1.00 → semua prediksi "Loyal" benar (tidak ada data Churn yang salah masuk ke Loyal).
- Recall = 1.00 → semua data Loyal aktual juga terdeteksi dengan benar.
- F1-score = 1.00 → sempurna.

Artinya, baik kelas Churn maupun Loyal diprediksi dengan akurasi 100% tanpa kesalahan sama sekali.

Overall Metrics

- Accuracy = 1.00 (100%): Dari total 6 data uji, semuanya diprediksi dengan benar.
- Macro avg & Weighted avg = 1.00 : Karena kedua kelas seimbang (3 Churn, 3 Loyal) dan performanya sempurna, semua nilai rata-rata juga 100%.

Confusion Matrix

- Kotak diagonal kuning (3 + 3) menunjukkan semua prediksi benar
- Tidak ada kotak non-diagonal terisi (tidak ada kesalahan klasifikasi).

Kesimpulan

- **Random Forest** berhasil memprediksi semua data dengan sempurna pada dataset kecil
- Hal ini menunjukkan **keunggulan ensemble method**: menggabungkan banyak pohon keputusan membuat model lebih stabil dan akurat dibanding Decision Tree tunggal.
- Namun, perlu diingat: hasil sempurna seperti ini bisa jadi dipengaruhi oleh dataset yang kecil dan sederhana. Pada dataset nyata yang lebih besar dan kompleks, akurasi biasanya tidak akan setinggi ini, tapi Random Forest tetap cenderung lebih konsisten daripada Decision Tree.

6. Perbandingan Singkat Antar Model

Bagian ini membandingkan performa KNN, Decision Tree, dan Random Forest berdasarkan metrik evaluasi (Accuracy, Precision, Recall, F1-Score) khusus untuk kelas Churn. Tujuannya agar mahasiswa memahami kelebihan dan kekurangan tiap algoritma. Hasil evaluasi ditampilkan dalam bentuk tabel agar mudah dianalisis.

```
# Import fungsi tambahan untuk menghitung precision, recall, dan f1-score
from sklearn.metrics import precision_recall_fscore_support

# Definisikan fungsi untuk meringkas metrik evaluasi
def summarize_metrics(y_true, y_pred, pos_label='Churn'):
    acc = accuracy_score(y_true, y_pred)  # Hitung akurasi model
    prec, rec, f1, _ = precision_recall_fscore_support(
        y_true, y_pred,                   # Data aktual dan hasil prediksi
        average='binary',                 # Fokus hanya pada satu kelas positif
        pos_label=pos_label,              # Kelas positif yang dipilih = 'Churn'
        zero_division=0                   # Jika ada pembagian nol, isi dengan 0
    )
    return acc, prec, rec, f1           # Kembalikan 4 metrik utama
```

```
# Siapkan list kosong untuk menyimpan ringkasan hasil tiap model
summary = []

# Lakukan evaluasi pada ketiga model yang sudah dibuat
for name, yhat in [
    ('KNN', y_pred_knn),                # Prediksi dari model KNN
    ('Decision Tree', y_pred_dt),       # Prediksi dari Decision Tree
    ('Random Forest', y_pred_rf)        # Prediksi dari Random Forest
]:
    # Hitung metrik untuk masing-masing model
    acc, prec, rec, f1 = summarize_metrics(y_test, yhat, pos_label='Churn')

    # Simpan hasilnya dalam bentuk dictionary lalu masukkan ke summary list
    summary.append({
        'Model': name,
        'Accuracy': round(acc, 3),          # Akurasi dibulatkan 3 digit
        'Precision (Churn)': round(prec, 3),
        'Recall (Churn)': round(rec, 3),
        'F1 (Churn)': round(f1, 3)
    })

# Ubah summary list menjadi DataFrame agar lebih rapi
df_summary = pd.DataFrame(summary)

# Cetak hasil ringkasan performa
print('📊 Ringkasan Performa Model (positif = kelas Churn)')
display(df_summary)   # Tampilkan tabel ringkasan di notebook
```

Setelah di jalanakn scrip tersebut akan menghasilkan output sebagai berikut:

```
📊 Ringkasan Performa Model (positif = kelas Churn)
```

	Model	Accuracy	Precision (Churn)	Recall (Churn)	F1 (Churn)
0	KNN	1.000	1.0	1.000	1.0
1	Decision Tree	0.833	1.0	0.667	0.8
2	Random Forest	1.000	1.0	1.000	1.0

Interpretasi Hasil Perbandingan Model

- **KNN:** Memberikan hasil sempurna (semua metrik = 1.00). Performa tinggi ini dipengaruhi oleh dataset kecil dan sederhana, serta penggunaan standarisasi yang tepat.
- **Decision Tree:** Akurasi 83% dan precision 1.00, namun recall hanya 0.667. Artinya, model gagal mengenali sebagian data *Churn*, menunjukkan potensi overfitting pada dataset kecil.
- **Random Forest:** Juga mencapai hasil sempurna. Keunggulannya berasal dari kombinasi banyak pohon, membuat model lebih stabil dan akurat.

Kesimpulan

KNN dan Random Forest adalah model terbaik untuk dataset ini karena mencapai hasil sempurna. Random Forest lebih direkomendasikan untuk penggunaan nyata karena lebih stabil terhadap variasi data, sedangkan KNN cocok untuk dataset kecil yang telah distandarisasi.

D. LATIHAN DAN TUGAS

1. LATIHAN

Untuk menguatkan pemahaman, lakukan latihan praktikum klasifikasi pada kasus dan dataset yang berbeda dari yang digunakan di bagian praktikum utama. Tujuan Latihan: Mahasiswa mampu membangun model klasifikasi dari nol, menguji beberapa algoritma, dan melakukan evaluasi hasil untuk memilih model terbaik.

a) Buat dataset dummy baru dengan minimal 50 baris data tentang status pembelian pelanggan e-commerce dengan struktur fitur seperti:
   - Umur (angka 18–60)
   - Pendapatan_juta (angka desimal 1–100)
   - Frekuensi_Kunjungan (angka 1–30)
   - Diskon_Favorit (0 = tidak suka diskon, 1 = suka diskon)
   - Status_Pembeli (label target: Aktif atau Pasif)

b) Simpan dataset dalam format CSV.

c) Bangun dan latih tiga model klasifikasi:
   - K-Nearest Neighbor (KNN)
   - Decision Tree
   - Random Forest

d) Evaluasi masing-masing model menggunakan:
   - Confusion Matrix
   - Classification Report (Akurasi, Precision, Recall, F1-Score)

e) Tampilkan dan bandingkan hasil evaluasi ketiga model dalam bentuk tabel sederhana.

f) Berikan analisis singkat (3–5 kalimat) mengenai:
   - Model mana yang paling akurat
   - Alasan kemungkinan perbedaan hasil antar model

2. TUGAS

Tujuan Tugas: Mahasiswa memahami konsep dasar klasifikasi, mengenal karakteristik tiga algoritma utama, serta memahami cara menilai performa model dengan tepat.

Jawablah pertanyaan berikut secara singkat namun jelas (gunakan bahasa sendiri):

a) Jelaskan perbedaan utama antara klasifikasi dan regresi dalam machine learning.

b) Berikan 3 contoh kasus nyata yang cocok diselesaikan dengan pendekatan klasifikasi.

c) Jelaskan secara sederhana cara kerja algoritma K-Nearest Neighbor (KNN).

d) Sebutkan kelebihan dan kelemahan Decision Tree dibandingkan KNN.

e) Mengapa Random Forest lebih tahan terhadap overfitting dibanding Decision Tree tunggal?

f) Jelaskan arti dari True Positive, False Positive, True Negative, dan False Negative pada confusion matrix.

g) Kapan kita sebaiknya lebih mengutamakan Recall daripada Precision, dan sebaliknya?

h) Apa makna nilai AUC yang mendekati 1.0 pada ROC Curve?

i) Mengapa penting bagi bisnis digital untuk dapat memprediksi pelanggan churn lebih awal?

j) Bagaimana menurut Anda, peran data berkualitas terhadap hasil model klasifikasi?

MODUL VI
CLUSTERING & SEGMENTASI PASAR

A. CAPAIAN PEMBELAJARAN

1) Mahasiswa memahami konsep clustering dan perbedaannya dengan klasifikasi.
2) Mahasiswa mengenal algoritma clustering populer (K-Means & Hierarchical Clustering).
3) Mahasiswa mampu menerapkan clustering untuk segmentasi pasar.
4) Mahasiswa dapat menginterpretasikan hasil clustering dalam konteks bisnis.

B. DASAR TEORI

1. Konsep Clustering

a) Clustering sebagai Metode *Unsupervised Learning*

Clustering adalah metode unsupervised learning yang mengelompokkan data tanpa label berdasarkan kemiripan antar fitur. Berbeda dengan supervised learning seperti klasifikasi, clustering mencari pola dalam data mentah untuk membentuk kelompok yang alami.

b) Tujuan Clustering: Mengelompokkan Data Berdasarkan Kemiripan

Tujuan utamanya adalah menyatukan data yang mirip dalam satu cluster, dan memisahkan data yang berbeda. Misalnya, dalam e-commerce, clustering bisa mengelompokkan pelanggan berdasarkan kebiasaan belanja atau mengidentifikasi produk yang sering dibeli bersamaan untuk strategi cross-selling.

c) Perbedaan Clustering vs Klasifikasi

Aspek	Clustering (Unsupervised)	Klasifikasi (Supervised)
Label data	Tidak ada label	Ada label target (kelas)
Tujuan	Menemukan pola & kelompok tersembunyi	Memprediksi label data baru
Cara kerja	Menghitung kemiripan antar data	Belajar pola dari data berlabel
Contoh kasus	Segmentasi pelanggan, analisis perilaku	Deteksi spam, churn vs loyal

Clustering cocok untuk eksplorasi data awal dan menemukan insight tersembunyi.

2. Algoritma Clustering

a) K-Means Clustering

K-Means mengelompokkan data ke dalam k cluster berdasarkan jarak ke titik pusat (centroid). Proses iteratif dilakukan hingga posisi centroid stabil.

- Kelebihan: Cepat, sederhana, cocok untuk data besar.
- Kekurangan: Perlu menentukan k di awal, sensitif terhadap outlier dan bentuk cluster.

b) Hierarchical Clustering

Hierarchical Clustering membentuk struktur pohon (dendrogram), tanpa perlu menentukan jumlah cluster di awal.

- Bottom-Up (Agglomerative): Mulai dari data individual lalu digabung.
- Top-Down (Divisive): Mulai dari satu cluster besar, lalu dipecah.

Kelebihan: Tidak butuh jumlah cluster dari awal, mudah divisualisasikan.
Kekurangan: Kurang efisien untuk data besar.

c) Menentukan Jumlah Cluster: Elbow & Silhouette

- Elbow Method: Melihat titik "siku" pada grafik WCSS untuk menentukan k optimal.
- Silhouette Score: Mengukur seberapa baik data cocok dalam clusternya (nilai mendekati 1 lebih baik).

3. Aplikasi Clustering dalam Bisnis Digital

a) Segmentasi Pelanggan: Clustering membantu membagi pelanggan e-commerce berdasarkan perilaku seperti frekuensi belanja, nilai transaksi, dan respons terhadap promo. Setiap segmen dapat diberi pendekatan pemasaran berbeda.

b) Analisis Pola Belanja: Digunakan untuk menemukan pola pembelian bersamaan, seperti pelanggan yang membeli popok juga membeli tisu basah. Ini mendukung bundling, rekomendasi produk, dan strategi katalog.

c) Market Segmentation: Clustering membantu membagi pasar menjadi segmen homogen berdasarkan demografi, minat, atau perilaku. Hal ini memungkinkan strategi pemasaran yang lebih personal dan efisien.

Kesimpulan: Clustering memungkinkan identifikasi pola tersembunyi dari data mentah. Dalam bisnis digital, ini berguna untuk memahami pelanggan dan mendukung pengambilan keputusan berbasis data.

C. PRAKTIKUM

Pada bagian ini, mahasiswa akan belajar menerapkan teknik clustering untuk melakukan segmentasi pelanggan e-commerce berdasarkan data transaksi sederhana. Mahasiswa akan menggunakan dua algoritma populer:

- K-Means Clustering
- Hierarchical Clustering

Selain itu, mahasiswa juga akan:

- menguji jumlah cluster berbeda (eksperimen *k*),
- memvisualisasikan hasil clustering dalam scatter plot berwarna, dan
- membuat dendrogram untuk melihat struktur cluster secara hierarkis.

1. Persiapan Dataset Pelanggan E-Commerce (Dummy)

- Buka https://colab.research.google.com
- Klik **File** → **New notebook**, beri nama:PraktikumML-Bab6_NIM_NamaLengkap.ipynb
- Buat script python seperti berikut dan jalankan untuk membuat Dataset Pelanggan E-Commerce (Dummy)

```
# --- Membuat dataset dummy pelanggan e-commerce ---
import pandas as pd
import numpy as np
# modul untuk mengunduh file dari Google Colab
from google.colab import files

# Atur seed random agar hasil random bisa direplikasi
np.random.seed(42)

# Membuat data dummy dengan 3 fitur:
data = {
    # nilai integer 1-30, total 50 sampel
    'Frekuensi_Transaksi': np.random.randint(1, 30, 50),
    # nilai float 50-500, dibulatkan 2 desimal
    'Rata2_Nilai_Transaksi': np.round(np.random.uniform(50, 500, 50), 2),
    # nilai integer 1-36 bulan
    'Lama_Berlangganan_bulan': np.random.randint(1, 36, 50)
}

# Konversi dictionary ke DataFrame agar lebih mudah dianalisis
df = pd.DataFrame(data)
# Simpan dataset ke file CSV agar bisa digunakan di eksperimen ML berikutnya
df.to_csv('data_pelanggan_ecommerce.csv', index=False)
```

```
# Tampilkan contoh 5 baris pertama
print('Contoh 5 baris pertama dataset:')
display(df.head())
# Mengecek apakah file CSV berhasil dibuat
!ls -l /content/*.csv

# Unduh file dataset ke lokal (opsional, untuk menyimpan di komputer Anda)
print('\n Mengunduh file dataset...')
files.download('data_pelanggan_ecommerce.csv')
```

Penjelasan fitur:

- Frekuensi_Transaksi → jumlah pembelian yang dilakukan pelanggan
- Rata2_Nilai_Transaksi → rata-rata nominal setiap transaksi (dalam ribuan rupiah)
- Lama_Berlangganan_bulan → lama waktu menjadi pelanggan

```
Contoh 5 baris pertama dataset:
```

	Frekuensi_Transaksi	Rata2_Nilai_Transaksi	Lama_Berlangganan_bulan
0	7	70.90	28
1	20	323.40	7
2	29	126.74	9
3	15	79.27	8
4	11	477.00	12

```
-rw-r--r-- 1 root root 677 Sep 24 12:53 /content/data_pelanggan_ecommerce.csv

 Mengunduh file dataset...
```

2. Standarisasi & K-Means Clustering (k=3)

Dalam pengolahan data, langkah penting yang perlu dilakukan sebelum menerapkan algoritma adalah standarisasi. Hal ini dikarenakan setiap fitur biasanya memiliki skala yang berbeda, misalnya frekuensi transaksi yang berkisar puluhan dibandingkan dengan nilai belanja yang bisa mencapai ratusan. Perbedaan skala tersebut dapat menimbulkan bias ketika dihitung jaraknya, sehingga perlu dinormalisasi agar semua fitur berada pada skala yang sama, yaitu dengan nilai rata-rata nol dan standar deviasi satu.

```
# Import alat untuk menstandarisasi fitur (mean=0, std=1)
from sklearn.preprocessing import StandardScaler
# Import algoritma K-Means untuk clustering
from sklearn.cluster import KMeans
# Import matplotlib untuk visualisasi
import matplotlib.pyplot as plt

# Buat objek scaler
scaler = StandardScaler()
# Fit ke data & transform semua fitur → skala seragam
X_scaled = scaler.fit_transform(df)
# Inisialisasi K-Means dengan 3 cluster (k=3)
kmeans = KMeans(n_clusters=3, random_state=42)
# Latih K-Means pada data terskalakan & ambil label cluster
labels_k3 = kmeans.fit_predict(X_scaled)
# Simpan label cluster ke kolom baru pada DataFrame
df['Cluster_K3'] = labels_k3

# Lihat beberapa baris pertama (cek kolom 'Cluster_K3')
display(df.head())

# Visualisasi scatter (2 fitur utama)
plt.figure()      # Siapkan canvas gambar
plt.scatter(      # Plot sebaran data 2D
```

```
    df['Frekuensi_Transaksi'],    # Sumbu-X: Frekuensi transaksi
    df['Rata2_Nilai_Transaksi'],  # Sumbu-Y: Rata-rata nilai transaksi
    c=df['Cluster_K3'],   # Warna titik berdasarkan label cluster hasil K-Means
    s=60        # Ukuran marker
)
plt.xlabel('Frekuensi Transaksi')           # Label sumbu-X
plt.ylabel('Rata-rata Nilai Transaksi')     # Label sumbu-Y
plt.title('Hasil Clustering K-Means (k=3)') # Judul plot
plt.colorbar(label='Cluster')               # Legenda warna cluster
plt.grid(True)                              # Grid bantu pembacaan
plt.show()                                  # Tampilkan plot ke layar
```

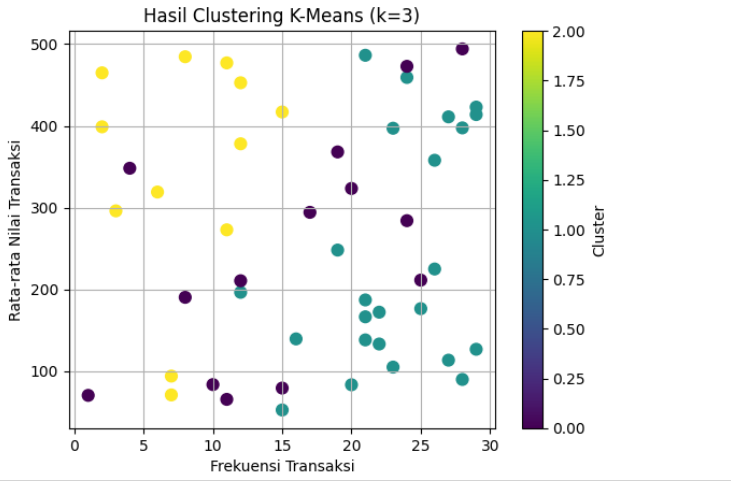
Penjelasan:

- K-Means akan membagi data menjadi 3 kelompok berdasarkan kemiripan
- Warna titik pada scatter plot menunjukkan cluster masing-masing pelanggan

Setelah data distandarisasi, algoritma **K-Means Clustering** dapat dijalankan. Metode ini bekerja dengan cara membagi data ke dalam sejumlah kelompok yang sudah ditentukan sebelumnya, misalnya tiga cluster. Prosesnya dimulai dengan memilih titik pusat atau *centroid*, lalu setiap data akan dikelompokkan berdasarkan jarak terdekat dengan centroid tersebut. Proses ini dilakukan berulang kali hingga posisi centroid stabil dan tidak lagi banyak berubah.

Hasil akhir dari proses tersebut terlihat pada dataset yang kini memiliki kolom tambahan berisi label cluster. Misalnya, pada contoh ini terbentuk tiga kelompok pelanggan yang ditandai dengan angka 0, 1, dan 2. Untuk memvisualisasikannya, scatter plot dapat digunakan, di mana warna berbeda menunjukkan segmen pelanggan yang berbeda pula, seperti kelompok pembeli hemat, pelanggan dengan frekuensi tinggi, hingga pelanggan premium dengan nilai belanja lebih besar.

	Frekuensi_Transaksi	Rata2_Nilai_Transaksi	Lama_Berlangganan_bulan	Cluster_K3
0	7	70.90	28	2
1	20	323.40	7	0
2	29	126.74	9	1
3	15	79.27	8	0
4	11	477.00	12	2



Lakukan Refleksi Singkat

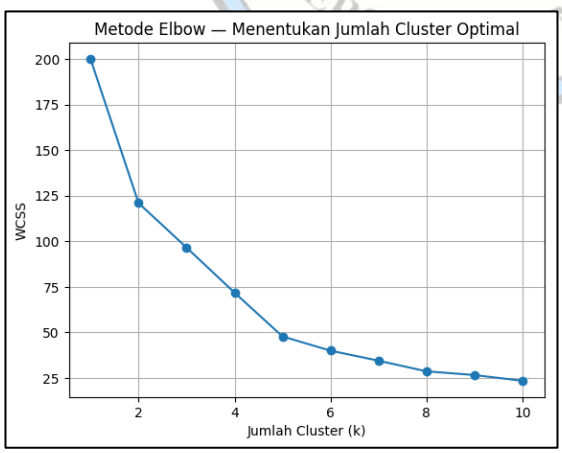
- Apakah sebaran tiap cluster terlihat jelas terpisah?
- Coba ganti `n_clusters` (misal 2, 4, 5) dan amati perubahan pola.

3. Menentukan Jumlah Cluster — Metode Elbow

```
# --- Metode Elbow untuk menentukan jumlah cluster optimal ---
wcss = []  # list untuk menyimpan nilai WCSS (Within-Cluster Sum of Squares)

# Uji jumlah cluster dari 1 sampai 10
for k in range(1, 11):
    # buat model K-Means dengan k cluster
    km = KMeans(n_clusters=k, random_state=42)
    km.fit(X_scaled)          # latih model pada data yang sudah distandarisasi
    wcss.append(km.inertia_) # simpan nilai inertia (WCSS) dari model

# Visualisasi grafik Elbow
plt.figure()
plt.plot(range(1, 11), wcss, marker='o')  # sumbu X = jumlah cluster, Y = WCSS
plt.title('Metode Elbow – Menentukan Jumlah Cluster Optimal')
plt.xlabel('Jumlah Cluster (k)')
plt.ylabel('WCSS')
plt.grid(True)
plt.show()
```

Deskripsi:

- WCSS (Within-Cluster Sum of Squares): Nilai ini menunjukkan seberapa rapat data dalam tiap cluster terhadap centroid-nya. Semakin kecil WCSS, semakin baik kualitas pengelompokan.
- Metode Elbow: Untuk menentukan jumlah cluster terbaik, dilakukan perhitungan WCSS untuk berbagai nilai k (misalnya 1–10). Hasilnya divisualisasikan dalam bentuk grafik.
- Interpretasi grafik: Pada grafik, biasanya terlihat bahwa penurunan WCSS tajam di awal, lalu melambat setelah titik tertentu. Titik belokan atau “elbow” inilah yang dianggap sebagai jumlah cluster optimal.



Interpretasi Hasil Analisis (Grafik Elbow)

Grafik Elbow menunjukkan bahwa nilai **WCSS (Within-Cluster Sum of Squares)** turun tajam dari k=1 hingga k=2, kemudian terus menurun cukup konsisten sampai k=5. Hal ini menandakan bahwa penambahan cluster pada tahap awal memberikan peningkatan signifikan dalam kualitas pemisahan data. Jika diperhatikan, terdapat dua kemungkinan titik belokan yang bisa dianggap sebagai "elbow". Pertama, pada **k=3**, di mana laju penurunan mulai melandai dibandingkan dua titik sebelumnya. Kedua, pada **k=5**, karena setelah titik ini penurunan WCSS semakin kecil (hanya sekitar 10–12 poin), sehingga tambahan cluster tidak lagi memberikan perbaikan besar.

Dengan demikian, baik k=3 maupun k=5 dapat dipertimbangkan sebagai jumlah cluster yang optimal. Pemilihan titik mana yang digunakan sangat bergantung pada tujuan analisis. Jika ingin segmen pelanggan yang sederhana dan mudah dipahami, **k=3** sudah cukup. Namun, jika tujuan analisis adalah membedakan pelanggan dengan lebih detail dan granular, maka **k=5** bisa menjadi pilihan yang lebih tepat.

4. Eksperimen Silhouette Score

Silhouette Score adalah ukuran kualitas clustering yang memperhitungkan kekompakan data dalam cluster dan pemisahan antar-cluster.Nilainya berkisar antara -1 sampai 1:

- Mendekati 1 → cluster terpisah dengan baik dan data di dalam cluster kompak.
- Mendekati 0 → cluster saling tumpang tindih.
- Negatif → data salah kelompok (lebih dekat ke cluster lain).

Dengan menguji berbagai nilai k, kita bisa membandingkan kualitas cluster dan memilih jumlah cluster terbaik berdasarkan skor tertinggi.

```python
# Import metric untuk mengukur kualitas clustering
from sklearn.metrics import silhouette_score

sil_scores = []  # list untuk menyimpan hasil skor

# Uji jumlah cluster mulai dari k=2 sampai k=10
for k in range(2, 11):
    # buat model K-Means dengan k cluster
    km = KMeans(n_clusters=k, random_state=42)
    # latih model dan prediksi label cluster
    labels = km.fit_predict(X_scaled)
    # hitung Silhouette Score untuk hasil cluster
    sil = silhouette_score(X_scaled, labels)
    # simpan hasil (jumlah cluster, skor)
    sil_scores.append((k, sil))

# Cetak hasil skor setiap k
print('k  |  Silhouette Score')
for k, s in sil_scores:
    print(f'{k:<2} | {s:.3f}')
```

```
k  |  Silhouette Score
2  | 0.365
3  | 0.347
4  | 0.376
5  | 0.441
6  | 0.432
7  | 0.386
8  | 0.399
9  | 0.379
10 | 0.380
```

Interpretasi Berdasarkan hasil perhitungan Silhouette Score:

- Nilai tertinggi diperoleh pada **k=5** dengan skor **0.441**, disusul oleh **k=6** dengan skor **0.432**.
- Skor pada **k=2** hingga **k=4** relatif lebih rendah (0.365–0.376), menunjukkan kualitas pemisahan yang tidak sebaik pada k=5.
- Mulai dari **k=7 hingga k=10**, skor cenderung menurun dan mendekati 0.38–0.39, artinya menambah cluster lebih banyak tidak memberikan peningkatan signifikan.

Dari hasil ini, dapat disimpulkan bahwa **jumlah cluster optimal adalah k=5**, karena menghasilkan skor Silhouette tertinggi dan menunjukkan keseimbangan terbaik antara kekompakan internal cluster dan jarak antar cluster.

- **Dari Metode Elbow:** kita melihat bahwa setelah k=5, penurunan WCSS mulai jauh lebih kecil (hanya 10–12 poin), sedangkan dari k=2 → k=5 penurunannya masih konsisten sekitar 25 poin. Artinya, "siku" yang jelas muncul di sekitar **k=5**.
- **Dari Silhouette Score:** Hasil perhitungan menunjukkan bahwa skor tertinggi juga berada pada **k=5** (0.441). Ini artinya kualitas clustering pada k=5 lebih baik dibanding jumlah cluster lain, termasuk k=3 yang sebelumnya dianggap sebagai elbow pertama.
- **Kesimpulan gabungan:** Dengan demikian, analisis **Elbow** dan **Silhouette Score** sama-sama mendukung bahwa **k=5 adalah jumlah cluster yang optimal** untuk dataset ini. Elbow menunjukkan bahwa setelah k=5, penurunan WCSS mulai melandai, sedangkan Silhouette Score memberikan bukti kuantitatif bahwa k=5 menghasilkan cluster paling kompak dan terpisah dengan baik.

5. Implementasi Hierarchical Clustering & Dendrogram

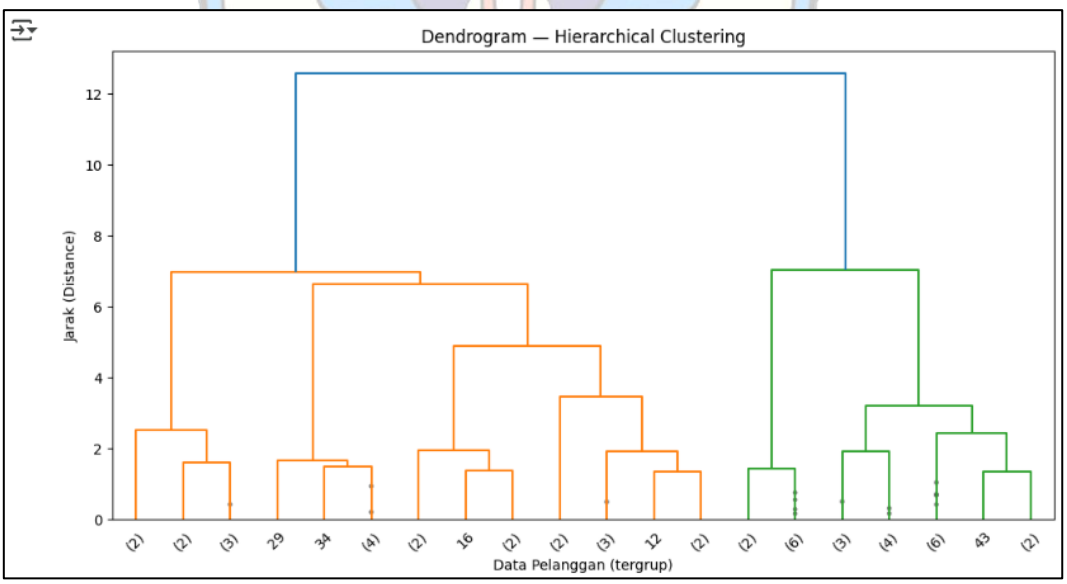
Hierarchical Clustering membentuk pohon cluster (dendrogram), Kita "memotong" pohon untuk mengambil jumlah cluster tertentu (contoh: 3 cluster)

```
# mengimpor tiga fungsi utama dari modul scipy.cluster.hierarchy
from scipy.cluster.hierarchy import linkage, dendrogram, fcluster
# linkage = membuat struktur penggabungan cluster.
# dendrogram = menggambarkan struktur tersebut dalam bentuk pohon.
# fcluster = memotong pohon untuk mengambil hasil cluster akhir (3 cluster).
import matplotlib.pyplot as plt

# --- 1. Bangun Linkage Matrix ---
# Linkage = cara mengukur jarak antar data/cluster dlm hierarchical clustering
# Method 'ward' digunakan untuk meminimalkan variasi dlm cluster (data numerik)
Z = linkage(X_scaled, method='ward')

# --- 2. Plot Dendrogram ---
plt.figure(figsize=(12, 6))
dendrogram(
    Z,
    truncate_mode='lastp',  # hanya tampilkan bagian akhir (kelompok terakhir)
    p=20,                   # tampilkan 20 grup data terakhir
    leaf_rotation=45,       # rotasi label sumbu X
    leaf_font_size=10,      # ukuran font label
    show_contracted=True    # tampilkan cluster yang diringkas
)
plt.title('Dendrogram – Hierarchical Clustering')
plt.xlabel('Data Pelanggan (tergrup)')
plt.ylabel('Jarak (Distance)')
plt.show()

# --- 3. Membentuk Cluster ---
# Potong dendrogram menjadi 3 cluster
labels_hc = fcluster(Z, t=3, criterion='maxclust')
# Simpan label cluster ke dataframe
df['Cluster_HC3'] = labels_hc

# --- 4. Visualisasi hasil clustering ---
plt.figure()
for cl in sorted(df['Cluster_HC3'].unique()):
    sub = df[df['Cluster_HC3'] == cl]
```

```
        plt.scatter(
            sub['Frekuensi_Transaksi'],
            sub['Rata2_Nilai_Transaksi'],
            s=60, label=f'Cluster {cl}'
        )
    plt.title('Hasil Hierarchical Clustering (3 cluster)')
    plt.xlabel('Frekuensi Transaksi')
    plt.ylabel('Rata-rata Nilai Transaksi')
    plt.legend()
    plt.grid(True)
    plt.show()
```

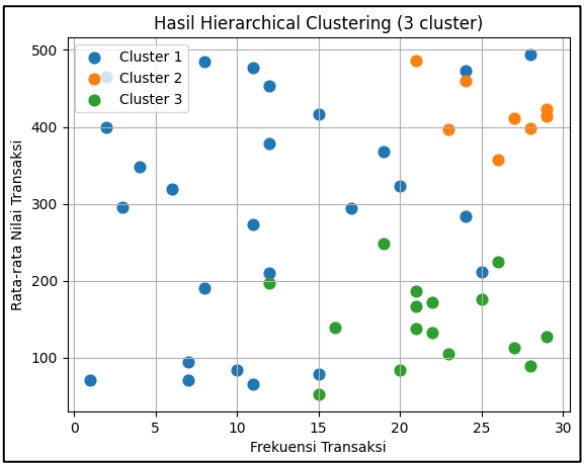
Hierarchical Clustering adalah metode pengelompokan data yang menyusun struktur bertahap/bertingkat, mirip seperti pohon keluarga. Dengan algoritma ini, setiap data awalnya dianggap sebagai cluster terpisah, lalu digabungkan secara bertahap berdasarkan tingkat kemiripan sampai terbentuk satu cluster besar.

- Linkage (Ward Method): digunakan untuk menggabungkan cluster dengan cara meminimalkan variasi (variance) di dalam cluster, sehingga hasil kelompok lebih kompak.
- Dendrogram: visualisasi berbentuk pohon yang memperlihatkan proses penggabungan cluster. Dari dendrogram, kita bisa memilih titik potong (cut) untuk menentukan jumlah cluster tertentu, misalnya 3.
- Hasil: pada dataset ini, dendrogram dipotong menjadi 3 cluster, kemudian hasilnya divisualisasikan dalam scatter plot menggunakan dua fitur utama (Frekuensi Transaksi dan Rata-rata Nilai Transaksi).



Dendrogram memperlihatkan proses penggabungan cluster berdasarkan jarak (similaritas).

- Sumbu Y menunjukkan **jarak (distance)**, artinya semakin tinggi percabangan, semakin jauh perbedaan antar cluster.
- Dari dendrogram terlihat bahwa pemotongan pada jarak sekitar **6–7** menghasilkan **3 cluster utama**, yang sesuai dengan hasil scatter plot.
- Struktur bertingkat ini menunjukkan bahwa cluster 2 (oranye) dan cluster 3 (hijau) cukup jelas terpisah, sedangkan cluster 1 (biru) memiliki variasi data yang lebih tersebar.



Kesimpulan

- Cluster 1 (biru): Pelanggan tidak rutin, kadang bertransaksi kecil kadang besar → "pelanggan acak".
- Cluster 2 (oranye): Pelanggan loyal bernilai tinggi → sangat penting untuk dipertahankan.
- Cluster 3 (hijau): Pelanggan rutin, tetapi dengan transaksi kecil → berpotensi ditingkatkan nilainya melalui promosi/upselling.

Hierarchical Clustering berhasil memvisualisasikan perbedaan segmen pelanggan, sehingga perusahaan bisa menyusun strategi pemasaran yang berbeda untuk tiap cluster.

D. LATIHAN DAN TUGAS

Setelah menjalankan kedua metode praaktikum di atas Bandingkan hasil cluster K-Means vs Hierarchical, Lihat apakah kelompok pelanggan yang terbentuk memiliki karakteristik berbeda (misalnya cluster dengan transaksi tinggi vs rendah) lalu Jelaskan potensi strategi pemasaran untuk setiap segmen

1. Latihan

Lakukan praktik berikut dengan **dataset dummy yang telah dibuat** pada praktikum, atau buat dataset baru dengan variasi sesuai instruksi.

a) **Latihan 1 — Variasi Jumlah Cluster K-Means**
- Gunakan dataset pelanggan e-commerce yang sudah ada.
- Jalankan algoritma **K-Means** dengan jumlah cluster = 2, 4, dan 5.
- Bandingkan hasil visualisasi scatter plot dengan hasil sebelumnya (k=3).
- Catat perbedaan pola segmentasi yang terbentuk.

b) **Latihan 2 — Analisis Silhouette Score**
- Hitung **Silhouette Score** untuk berbagai nilai k (misalnya k=2 hingga k=8).
- Tentukan nilai k terbaik berdasarkan skor tertinggi.
- Bandingkan hasilnya dengan metode Elbow.

c) **Latihan 3 — Hierarchical Clustering**
- Terapkan **Hierarchical Clustering** pada dataset yang sama.
- Tampilkan dendrogram dan ambil hasil cluster untuk k=3.
- Bandingkan hasil pengelompokan dengan K-Means. Apakah mirip atau berbeda?

2. Tugas

Jawablah pertanyaan berikut secara singkat dan jelas.

a) **Konsep Clustering**

- Apa perbedaan utama antara *clustering* (unsupervised learning) dan *klasifikasi* (supervised learning)?
- Mengapa standarisasi data diperlukan sebelum menggunakan K-Means?

b) **Algoritma Clustering**
- Jelaskan secara singkat cara kerja algoritma **K-Means**.
- Sebutkan kelebihan dan kekurangan **Hierarchical Clustering** dibanding K-Means.

c) **Evaluasi & Aplikasi**
- Apa fungsi **Metode Elbow** dan **Silhouette Score** dalam clustering?
- Berikan contoh **strategi pemasaran** berdasarkan hasil segmentasi pelanggan dengan clustering (misalnya cluster pelanggan loyal vs cluster pelanggan sensitif harga).

FAKULTAS EKONOMI
UNIVERSITAS KUNINGAN

MODUL VII
DATA PREPROCESSING & FEATURE ENGINEERING

A. CAPAIAN PEMBELAJARAN

1) Mahasiswa memahami pentingnya preprocessing sebelum membangun model ML.
2) Mahasiswa mampu melakukan data cleaning & penanganan missing values.
3) Mahasiswa mengenal normalisasi & standarisasi data.
4) Mahasiswa memahami konsep feature selection & feature extraction.

B. DASAR TEORI

1. Data Cleaning

a. Identifikasi Data Kotor

Langkah awal dalam membangun model Machine Learning adalah memastikan bahwa data yang digunakan bersih dan berkualitas. Data yang bermasalah dapat menghasilkan analisis yang menyesatkan dan model yang tidak akurat. Beberapa jenis data kotor yang umum dijumpai antara lain: duplikasi, yaitu data yang tercatat lebih dari satu kali akibat kesalahan input; outlier, yakni nilai ekstrem yang menyimpang dari pola umum dan berpotensi mengganggu model meskipun tidak selalu salah; serta error input, seperti kesalahan ketik atau data tidak masuk akal (contohnya usia 250 tahun atau nilai transaksi negatif). Mengenali dan memahami berbagai bentuk ketidaksesuaian data ini menjadi fondasi penting sebelum melanjutkan ke tahap analisis atau pemodelan lanjutan.

b. Penanganan Missing Values

Nilai yang hilang atau missing values sering kali muncul akibat kesalahan teknis, kelalaian pengisian data, atau responden yang tidak memberikan jawaban. Untuk mengatasinya, beberapa teknik umum dapat diterapkan. Mean/median imputation menggantikan nilai kosong dengan rata-rata atau median, tergantung distribusi data dan keberadaan outlier. Jika jumlah nilai hilang terlalu banyak, maka penghapusan baris/kolom (drop rows/columns) menjadi opsi yang lebih efisien. Selain itu, custom imputation juga sering digunakan untuk data kategorikal, seperti mengganti nilai kosong dengan "Unknown" atau label serupa. Pemilihan metode bergantung pada konteks, jenis data, serta tujuan analisis.

c. Contoh Kasus: Data Penjualan Online

Misalnya, dalam dataset transaksi toko online yang mencakup ID pelanggan, tanggal transaksi, jumlah barang, dan total pembelian, ditemukan beberapa permasalahan. Kolom Jumlah_Barang kosong karena kesalahan sistem, sementara Total_Pembelian mencatat angka negatif yang jelas merupakan kesalahan input. Selain itu, terdapat entri transaksi yang terduplikasi dengan nomor invoice yang sama. Untuk membersihkan data, langkah-langkah yang diambil meliputi: menghapus duplikasi berdasarkan invoice, mengganti nilai kosong menggunakan median, serta memperbaiki atau menghapus entri yang salah. Proses ini menjadikan dataset lebih konsisten dan siap digunakan untuk tahap transformasi lebih lanjut, seperti normalisasi dan rekayasa fitur.

2. Normalisasi & Standarisasi

Salah satu tahapan penting dalam data preprocessing adalah menyamakan skala antar fitur. Proses ini krusial karena banyak algoritma Machine Learning, seperti K-Nearest Neighbors (KNN) atau regresi linear, sangat sensitif terhadap perbedaan skala. Sebagai ilustrasi, jika sebuah dataset memiliki dua fitur — pendapatan dan usia — dan keduanya digunakan tanpa penyesuaian skala, maka fitur dengan rentang nilai lebih besar akan

mendominasi proses perhitungan jarak atau bobot model. Untuk mengatasi masalah ini, dua teknik utama digunakan: normalisasi dan standarisasi.

- **Normalisasi (Min-Max Scaling)**

  Normalisasi mengubah data agar berada pada rentang tertentu, biasanya **0 hingga 1**. Rumusnya sederhana:

$$x' = \frac{x - x_{min}}{x_{max} - x_{min}}$$

  Dengan cara ini, nilai terkecil dalam data akan menjadi 0, sedangkan nilai terbesar menjadi 1. Cocok digunakan ketika distribusi data tidak terlalu ekstrem dan kita ingin menjaga proporsi asli antar nilai.

- **Standarisasi (Z-Score Scaling)**

  Standarisasi mengubah data sehingga memiliki rata-rata (mean) = 0 dan standar deviasi = 1. Rumusnya adalah:

$$z = \frac{x - \mu}{\sigma}$$

  Teknik ini lebih kuat ketika data memiliki distribusi yang bervariasi atau terdapat outlier, karena fokusnya pada penyebaran data relatif terhadap rata-rata.

  Dengan kata lain, normalisasi menekan nilai data ke dalam skala tertentu, sedangkan standarisasi menyesuaikan data agar seimbang berdasarkan rata-rata dan sebarannya. Sebagai contoh, pada algoritma **KNN**, jarak antar titik data dihitung menggunakan metrik seperti Euclidean Distance. Tanpa skala yang seragam, fitur dengan rentang besar akan mendominasi perhitungan jarak. Demikian pula, dalam regresi linear, koefisien model bisa menjadi bias jika skala fitur tidak diseimbangkan terlebih dahulu.

3. Feature Selection & Feature Extraction

Kualitas model tidak hanya ditentukan oleh banyaknya data, tetapi juga oleh fitur yang digunakan. Feature selection dan feature extraction adalah dua strategi penting untuk meningkatkan efektivitas model. Ibarat bahan dalam resep, fitur yang tepat akan menghasilkan model yang akurat dan efisien.

Feature Selection (Pemilihan Fitur Relevan)

Proses ini bertujuan untuk menyaring fitur yang memiliki kontribusi signifikan terhadap output model, sambil mengabaikan fitur yang bersifat redundan atau mengandung noise. Teknik yang digunakan meliputi filter methods seperti uji korelasi atau chi-square, wrapper methods seperti recursive feature elimination, dan embedded methods seperti Lasso Regression yang secara otomatis mengecilkan koefisien fitur tidak penting. Dengan menyederhanakan jumlah fitur, proses pelatihan menjadi lebih cepat dan hasil model lebih stabil.

Feature Extraction (Transformasi Fitur)

Berbeda dari seleksi fitur, feature extraction mengubah fitur asli menjadi representasi baru yang lebih informatif. Salah satu teknik paling umum adalah Principal Component Analysis (PCA) yang mereduksi dimensi data dengan menciptakan fitur baru berdasarkan kombinasi linear dari fitur yang ada. Misalnya, dari 50 fitur, PCA dapat merangkum informasi utama ke dalam 10 fitur saja. Transformasi ini tidak hanya menyederhanakan data, tetapi juga membantu mengurangi overfitting dan mempermudah interpretasi hasil model.

.

4. Best Practices dalam Preprocessing

Preprocessing merupakan tahap krusial dalam pipeline Machine Learning, karena kualitas input akan sangat menentukan keluaran model. Pepatah “garbage in, garbage out” menjadi pengingat bahwa data yang buruk tidak dapat menghasilkan model yang baik, sebaik apapun algoritmanya.

Workflow Umum Preprocessing

Agar proses lebih efisien dan minim kesalahan, preprocessing idealnya mengikuti alur kerja terstruktur:

a) data cleaning untuk menangani duplikasi, missing values, dan error;
b) data transformation seperti normalisasi dan standarisasi;
c) feature engineering melalui pemilihan dan pembentukan fitur;
d) encoding data kategorikal dengan label encoding atau one-hot encoding;
e) splitting dataset menjadi data latih dan uji; serta
f) pembuatan pipeline preprocessing untuk menyatukan seluruh tahapan dalam satu alur otomatis.

Tools Pendukung Preprocessing di Scikit-Learn

Library scikit-learn menyediakan berbagai alat yang memudahkan preprocessing, antara lain: SimpleImputer untuk imputasi nilai hilang, MinMaxScaler dan StandardScaler untuk transformasi skala, LabelEncoder dan OneHotEncoder untuk konversi data kategorikal, SelectKBest untuk seleksi fitur, serta PCA untuk ekstraksi fitur. Semua komponen ini dapat dirangkai dalam Pipeline, memungkinkan proses yang konsisten dan dapat diulang dengan hasil serupa di berbagai skenario. Dengan pendekatan ini, preprocessing menjadi bagian yang tidak hanya penting, tetapi juga efisien dan reproducible dalam workflow data science modern.

C. PRAKTIKUM

Tujuan Praktikum:

a) Melakukan data cleaning (duplikasi, missing values, error input/outlier sederhana).
b) Menerapkan scaling (Min-Max & StandardScaler).
c) Menerapkan feature engineering: feature selection (SelectKBest) & feature extraction (PCA).
d) Membandingkan performa model sederhana (KNN) sebelum vs sesudah preprocessing.

Catatan: Kita gunakan dummy dataset penjualan online agar sesuai konteks modul.

1. Setup & Library

Pada tahap awal praktikum ini, kita menyiapkan semua library yang akan dipakai untuk data preprocessing, feature engineering, modeling, dan evaluasi.

- NumPy & Pandas → dasar pengolahan data.
- Matplotlib → visualisasi data.
- Scikit-learn (sklearn) → menyediakan tools untuk preprocessing (scaling, encoding, imputasi), pemilihan fitur, reduksi dimensi, hingga algoritma machine learning (KNN pada contoh ini).
- np.random.seed(42) → digunakan agar proses random (seperti pembagian data) konsisten setiap kali dijalankan.

Dengan setup ini, kita sudah siap masuk ke tahap berikutnya: memproses data mentah agar siap dipakai model ML.

```python
# Import library numerik & manipulasi data
import numpy as np
import pandas as pd

# Visualisasi
import matplotlib.pyplot as plt # Digunakan untuk membuat visualisasi data

# Preprocessing
# Membagi dataset menjadi data latih (train) dan data uji (test)
from sklearn.model_selection import train_test_split
# Untuk menangani missing value (nilai kosong) dengan metode tertentu (mean, median, mode)
from sklearn.impute import SimpleImputer
from sklearn.preprocessing import MinMaxScaler, StandardScaler, OneHotEncoder
# MinMaxScaler: normalisasi data ke rentang tertentu (biasanya 0-1)
# StandardScaler: standarisasi data agar memiliki mean=0 dan std=1
# OneHotEncoder: mengubah data kategori menjadi format numerik biner (dummy variable)

# Untuk menerapkan preprocessing berbeda pada kolom berbeda
from sklearn.compose import ColumnTransformer
# Untuk membuat alur preprocessing + modeling secara terstruktur
from sklearn.pipeline import Pipeline
from sklearn.feature_selection import SelectKBest, f_classif
# SelectKBest: memilih fitur terbaik berdasarkan skor tertentu
# f_classif: metode seleksi fitur menggunakan uji ANOVA F-value

# Principal Component Analysis: reduksi dimensi fitur
from sklearn.decomposition import PCA

# Model & Evaluasi
from sklearn.neighbors import KNeighborsClassifier
from sklearn.metrics import accuracy_score, classification_report, confusion_matrix
# accuracy_score: menghitung akurasi prediksi
# classification_report: laporan metrik evaluasi (precision, recall, f1-score)
# confusion_matrix: tabel prediksi vs aktual

# Agar hasil percobaan dapat direproduksi (tidak berubah-ubah)
np.random.seed(42)

print("Library siap dipakai.")   # Menandakan setup library sudah berhasil
```

```
Library siap dipakai.
```

2. Generate Dummy Dataset Penjualan Online

Kode ini membuat dataset dummy pelanggan toko online sebanyak 600 data.

- Variabel input meliputi kota, jumlah transaksi, nilai belanja rata-rata, diskon rata-rata, dan recency (hari sejak transaksi terakhir).
- Target output adalah variabel Loyal (1 = loyal, 0 = tidak loyal), dihitung menggunakan model logit sederhana + noise agar data lebih realistis.
- Dataset ini nantinya bisa digunakan untuk latihan klasifikasi (ML/AI).

```python
# Jumlah data yang akan dibuat
n = 600

# Membuat dataset dummy dengan Pandas DataFrame
data = pd.DataFrame({
    "ID_Pelanggan": np.arange(1000, 1000+n),   # ID unik pelanggan mulai dari 1000
    "Kota": np.random.choice(                  # Kota pelanggan dipilih secara acak
        ["Bandung", "Jakarta", "Surabaya", "Yogyakarta"],
        size=n,
        p=[0.25, 0.35, 0.25, 0.15]             # Probabilitas distribusi kota
    ),
    "Jumlah_Transaksi": np.random.poisson(lam=8, size=n),
    # Jumlah transaksi per pelanggan (distribusi Poisson rata-rata 8 transaksi)

    "Rata2_Nilai_Beli": np.random.normal(loc=250_000, scale=80_000, size=n),
    # Rata-rata nilai belanja (distribusi Normal, mean=250 ribu, std=80 ribu)
```

```
    "Diskon_Rata2": np.clip(
        np.random.normal(loc=0.15, scale=0.05, size=n), 0, 0.6
    ),
    # Diskon rata-rata (distribusi Normal, rata-rata 15%, dibatasi min=0, max=60%)

    "Hari_Semjak_Transaksi_Terakhir": np.random.randint(1, 180, size=n),
    # Recency (berapa hari sejak transaksi terakhir, antara 1-180 hari)
})

# Membuat target 'Loyal' dengan model logit sederhana
logit = (
    0.12 * data["Jumlah_Transaksi"] +                # Semakin banyak transaksi → lebih loyal
    0.000004 * data["Rata2_Nilai_Beli"] -            # Semakin tinggi nilai belanja → lebih loyal
    0.01 * data["Hari_Semjak_Transaksi_Terakhir"] +  # Semakin lama tidak belanja → kurang loyal
    0.3 * data["Diskon_Rata2"] +                     # Diskon mempengaruhi loyalitas
    np.random.normal(0, 0.5, size=n)                 # Tambahan noise/random agar lebih realistis
)

# Setelah membuat dataset "data"
data["Rata2_Nilai_Beli"] = data["Rata2_Nilai_Beli"].round(2)
data["Diskon_Rata2"] = data["Diskon_Rata2"].round(2)

# Mengubah logit menjadi probabilitas dengan fungsi sigmoid
prob = 1 / (1 + np.exp(-logit))

# Menentukan loyal atau tidak (threshold 0.5)
data["Loyal"] = (prob > 0.5).astype(int)

# Cek ukuran dataset dan tampilkan 5 baris pertama
print(data.shape)
data.head()
```

Output dari script tersebut setelah di eksekusi akan mengahasilkan dataset seperti berikut

```
(600, 7)
```

	ID_Pelanggan	Kota	Jumlah_Transaksi	Rata2_Nilai_Beli	Diskon_Rata2	Hari_Semjak_Transaksi_Terakhir	Loyal
0	1000	Surabaya	7	318029.54	0.15	43	1
1	1001	Surabaya	7	382263.71	0.19	17	1
2	1002	Jakarta	15	175560.97	0.12	102	1
3	1003	Bandung	5	89713.61	0.21	162	1
4	1004	Bandung	4	227809.14	0.16	111	0

3. Menyimpan Dataset (Google Colab)

Jika Anda ingin simpan ke Google Drive dan juga download langsung CSV, tambahkan kode berikut di bawahnya:

```
# Simpan dataset ke file CSV
filename = "dataset_penjualan_online.csv"
data.to_csv(filename, index=False)

# === Opsi 1: Simpan ke Google Drive ===
from google.colab import drive
drive.mount('/content/drive')
data.to_csv('/content/drive/My Drive/ColabNotebooks/Modul7/' + filename, index=False)
print("Dataset berhasil disimpan ke Google Drive.")

# === Opsi 2: Download langsung ===
from google.colab import files
files.download(filename)
```

```
Drive already mounted at /content/drive; to attempt to forcibly remount, call drive.mount(
Dataset berhasil disimpan ke Google Drive.
```

4. Sisipkan “Kekotoran Data” (duplikasi, missing, error input)

Kode ini menyisipkan kekotoran data ke dalam dataset:

- Duplikasi data → 10 baris digandakan untuk mensimulasikan data ganda.
- Missing values → beberapa nilai pada kolom numerik diubah menjadi kosong (NaN).
- Error input → dibuat nilai mustahil (negatif) dan outlier ekstrem (transaksi 999).

Dengan begini, dataset menjadi lebih realistis karena mencerminkan masalah nyata pada data dunia industri, yang harus dibersihkan sebelum dipakai machine learning.

```
# Membuat salinan dataset asli agar tidak mengubah data awal
dirty = data.copy()

# === Duplikasi data ===
# Pilih 10 baris secara acak dari dataset
dup_idx = np.random.choice(dirty.index, size=10, replace=False)

# Gandakan baris-barus tersebut, lalu gabungkan ke dataset
dirty = pd.concat([dirty, dirty.loc[dup_idx]], ignore_index=True)

# === Missing values ===
# Kosongkan sebagian nilai numerik di kolom tertentu
for col in ["Jumlah_Transaksi", "Rata2_Nilai_Beli"]:
    # ambil 20 baris acak
    mv_idx = np.random.choice(dirty.index, size=20, replace=False)
    dirty.loc[mv_idx, col] = np.nan    # isi dengan NaN (hilang)

# === Error input ===
# Kasus 1: Nilai mustahil (negatif untuk belanja)
err_idx1 = np.random.choice(dirty.index, size=3, replace=False)
dirty.loc[err_idx1, "Rata2_Nilai_Beli"] = -50000  # nilai mustahil (negatif)

# Kasus 2: Outlier ekstrem (jumlah transaksi terlalu besar)
err_idx2 = np.random.choice(dirty.index, size=3, replace=False)
dirty.loc[err_idx2, "Jumlah_Transaksi"] = 999     # outlier ekstrem

# Cek ukuran dataset setelah ditambah kekotoran
print("Ukuran data kotor:", dirty.shape)

# Tampilkan 5 sampel acak untuk verifikasi
dirty.sample(5)
```

```
Ukuran data kotor: (610, 7)
```

	ID_Pelanggan	Kota	Jumlah_Transaksi	Rata2_Nilai_Beli	Diskon_Rata2	Hari_Semjak_Transaksi_Terakhir	Loyal
228	1228	Jakarta	9.0	238079.15	0.22	165	1
380	1380	Surabaya	14.0	300432.19	0.12	116	1
560	1560	Surabaya	7.0	217998.23	0.14	19	1
51	1051	Jakarta	13.0	504744.53	0.16	132	1
395	1395	Jakarta	9.0	199760.22	0.18	15	1

5. Menyimpan Dataset Kotor (Google Colab Drive & Download)

tujan dari tahapan ini adalah memastikan bahwa dataset dummy yg kita buat di awal telah terupdate dengan menambahkan beberapa data kotor, caranya Tambahkan kode berikut setelahnya:

```
# Simpan dataset kotor ke file CSV
dirty_filename = "dataset_penjualan_online_kotor.csv"
dirty.to_csv(dirty_filename, index=False)

# === Opsi 1: Simpan ke Google Drive ===
from google.colab import drive
drive.mount('/content/drive')
dirty.to_csv('/content/drive/My Drive/ColabNotebooks/Modul7/' + dirty_filename, index=False)
print("Dataset kotor berhasil disimpan ke Google Drive.")

# === Opsi 2: Download langsung ke lokal ===
from google.colab import files
files.download(dirty_filename)
```

```
Drive already mounted at /content/drive; to attempt to forcibly remount, call drive.mount("/content/drive"
Dataset kotor berhasil disimpan ke Google Drive.
```

6. Data Cleaning (hapus duplikasi & Menangani nilai eror)

Tahap Data Cleaning ini memperbaiki kualitas dataset:

a) Menghapus duplikasi → baris yang sama berdasarkan kombinasi ID_Pelanggan + Kota + Jumlah Transaksi + Rata-rata Nilai Beli dihapus agar analisis tidak bias.

b) Menangani error input:
   - Nilai belanja negatif → diganti NaN (hilang).
   - Jumlah transaksi tidak wajar (>200) → juga diubah jadi NaN. → Alasannya: nilai-nilai ini dianggap "anomali" sehingga lebih aman diimputasi atau dibuang.

c) Cek missing values → menghitung jumlah data kosong (NaN) per kolom setelah proses cleaning.

```python
# Membuat salinan dataset kotor agar data asli tidak terpengaruh
clean = dirty.copy()

# === 3.1 Hapus Duplikasi ===
# Hitung jumlah baris sebelum pembersihan
before = clean.shape[0]

# Hapus baris duplikat berdasarkan kombinasi ID_Pelanggan + fitur inti perilaku
clean = clean.drop_duplicates(
    subset=["ID_Pelanggan", "Kota", "Jumlah_Transaksi", "Rata2_Nilai_Beli"]
)

# Hitung jumlah baris setelah pembersihan
after = clean.shape[0]

# Cetak berapa baris duplikat yang dihapus
print(f"Duplikasi dihapus: {before - after} baris")

# === 3.2 Tangani Error Input ===
# Kasus 1: Nilai belanja negatif (tidak masuk akal) → ubah menjadi NaN
clean.loc[clean["Rata2_Nilai_Beli"] < 0, "Rata2_Nilai_Beli"] = np.nan

# Kasus 2: Outlier ekstrem pada Jumlah_Transaksi (>200 transaksi) → ubah menjadi NaN
# Nantinya bisa ditangani (imputasi / dibuang)
clean.loc[clean["Jumlah_Transaksi"] > 200, "Jumlah_Transaksi"] = np.nan

# === 3.3 Cek Missing Values ===
print("Missing per kolom:")
print(clean.isna().sum())

# Tampilkan 5 baris pertama setelah cleaning
clean.head()
```

Output dari kode program tersebut akan tampil seperti berikut.

```
Duplikasi dihapus: 10 baris
Missing per kolom:
ID_Pelanggan                  0
Kota                          0
Jumlah_Transaksi             23
Rata2_Nilai_Beli             24
Diskon_Rata2                  0
Hari_Semjak_Transaksi_Terakhir    0
Loyal                         0
dtype: int64
```

	ID_Pelanggan	Kota	Jumlah_Transaksi	Rata2_Nilai_Beli	Diskon_Rata2	Hari_Semjak_Transaksi_Terakhir	Loyal
0	1000	Surabaya	7.0	318029.54	0.15	43	1
1	1001	Surabaya	7.0	382263.71	0.19	17	1
2	1002	Jakarta	15.0	175560.97	0.12	102	1
3	1003	Bandung	5.0	89713.61	0.21	162	1
4	1004	Bandung	NaN	227809.14	0.16	111	0

Interpretasi Hasil

a) Duplikasi dihapus: 10 baris → Artinya ada 10 data pelanggan yang ganda (sama persis di beberapa fitur inti) dan berhasil dibuang. Ini mencegah data bias (misalnya satu pelanggan dihitung dua kali).

b) Missing Values (hasil .isna().sum())
   - Jumlah_Transaksi: 23 nilai kosong
   - Rata2_Nilai_Beli: 24 nilai kosong
   - Kolom lain (ID_Pelanggan, Kota, Diskon_Rata2, Hari_Semjak_Transaksi_Terakhir, Loyal) = tidak ada missing → Missing ini muncul karena error input (nilai negatif / outlier ekstrem) kita set jadi NaN, ditambah memang ada missing yang kita sisipkan sebelumnya saat membuat dataset kotor.

Kesimpulan Sementara

- Dataset sudah lebih bersih: duplikasi hilang, error input sudah ditandai NaN.
- Namun, sekarang ada missing values (23 pada Jumlah_Transaksi dan 24 pada Rata2_Nilai_Beli) yang perlu ditangani (misalnya dengan imputasi mean/median atau metode lain).
- Dataset ini siap masuk ke tahap imputasi missing values agar bisa digunakan untuk pelatihan model machine learning.

7. Menyimpan Dataset (Google Colab)

tujan dari tahapan ini adalah memastikan bahwa dataset dummy yg kita buat di awal telah terupdate dengan membersihkan beberapa data kotor, caranya Tambahkan kode berikut setelahnya:

```
# Simpan dataset hasil cleaning ke CSV
clean_filename = "dataset_penjualan_online_clean.csv"
clean.to_csv(clean_filename, index=False)

# === Opsi 1: Simpan ke Google Drive ===
from google.colab import drive
drive.mount('/content/drive')
clean.to_csv('/content/drive/My Drive/ColabNotebooks/Modul7/' + clean_filename, index=False)
print("Dataset clean berhasil disimpan ke Google Drive.")

# === Opsi 2: Download otomatis ke lokal ===
from google.colab import files
files.download(clean_filename)
```

```
Drive already mounted at /content/drive; to attempt to forcibly remount, call drive.mount("/content/drive",
Dataset clean berhasil disimpan ke Google Drive.
```

8. Data Cleaning (imputasi missing values)

Dalam tahap Imputasi Missing Values ini Kita akan menggunakan SimpleImputer dari sklearn untuk mengisi nilai kosong (NaN) pada kolom numerik.

```
from sklearn.impute import SimpleImputer

# Membuat salinan agar lebih aman
imputed = clean.copy()

# Pilih kolom numerik yang mungkin ada missing values
num_cols = ["Jumlah_Transaksi", "Rata2_Nilai_Beli"]

# Buat objek SimpleImputer
# strategy="median" → mengganti NaN dengan nilai median kolom
imputer = SimpleImputer(strategy="median")
```

```
# Terapkan imputasi ke kolom numerik
imputed[num_cols] = imputer.fit_transform(imputed[num_cols])

# Cek ulang apakah masih ada missing values
print("Missing values setelah imputasi:")
print(imputed.isna().sum())

# Tampilkan 5 baris pertama untuk verifikasi
imputed.head()
```

```
Missing values setelah imputasi:
ID_Pelanggan                  0
Kota                          0
Jumlah_Transaksi              0
Rata2_Nilai_Beli              0
Diskon_Rata2                  0
Hari_Semjak_Transaksi_Terakhir    0
Loyal                         0
dtype: int64
```

	ID_Pelanggan	Kota	Jumlah_Transaksi	Rata2_Nilai_Beli	Diskon_Rata2	Hari_Semjak_Transaksi_Terakhir	Loyal
0	1000	Surabaya	7.0	318029.54	0.15	43	1
1	1001	Surabaya	7.0	382263.71	0.19	17	1
2	1002	Jakarta	15.0	175560.97	0.12	102	1
3	1003	Bandung	5.0	89713.61	0.21	162	1
4	1004	Bandung	8.0	227809.14	0.16	111	0

Deskripsi

- SimpleImputer → modul dari scikit-learn untuk mengisi data hilang.
- Kita pakai median (bukan mean) karena lebih robust terhadap outlier.
- Semua NaN di kolom Jumlah_Transaksi dan Rata2_Nilai_Beli akan diganti dengan nilai median kolom masing-masing.
- Setelah ini dataset tidak ada missing lagi → sudah siap untuk tahap scaling/encoding sebelum modeling.

9. Menyimpan Dataset (Google Colab)

```
# Simpan dataset hasil imputasi ke CSV
imputed_filename = "dataset_penjualan_online_imputed.csv"
imputed.to_csv(imputed_filename, index=False)

# === Opsi 1: Simpan ke Google Drive ===
from google.colab import drive
drive.mount('/content/drive')
imputed.to_csv('/content/drive/My Drive/ColabNotebooks/Modul7/' + imputed_filename, index=False)
print("Dataset imputed berhasil disimpan ke Google Drive.")

# === Opsi 2: Download otomatis ke lokal ===
from google.colab import files
files.download(imputed_filename)
```

```
Drive already mounted at /content/drive; to attempt to forcibly remount, call drive.mount("/content/drive",
Dataset imputed berhasil disimpan ke Google Drive.
```

10. Train/Test Split (hindari data leakage)

Pada tahap ini, dataset dipisahkan menjadi dua bagian: data latih (train) dan data uji (test) menggunakan fungsi train_test_split dari scikit-learn. Variabel target yang diprediksi adalah Loyal, sementara fitur yang digunakan untuk prediksi adalah semua kolom kecuali Loyal dan ID_Pelanggan (karena ID tidak relevan untuk pemodelan).

Proporsi pembagian data ditentukan sebesar 75% untuk training dan 25% untuk testing, dengan parameter stratify=y agar distribusi kelas target tetap seimbang di kedua subset. Selain itu, kolom fitur dipisahkan menjadi numerik (Jumlah_Transaksi, Rata2_Nilai_Beli, Diskon_Rata2, Hari_Semjak_Transaksi_Terakhir) dan kategori (Kota) untuk memudahkan proses preprocessing selanjutnya. Pembagian dataset seperti ini penting untuk menghindari data leakage, yaitu kondisi ketika informasi dari data uji secara tidak sengaja digunakan saat pelatihan, yang bisa membuat performa model tampak lebih baik dari sebenarnya.

```
# Pisahkan fitur (X) dan target (y)
# Fitur: semua kolom kecuali 'Loyal' (target) & 'ID_Pelanggan' (identifier)
X = clean.drop(columns=["Loyal", "ID_Pelanggan"])
# Target: variabel loyalitas pelanggan (0 = tidak loyal, 1 = loyal)
y = clean["Loyal"]

# Membagi dataset menjadi train (75%) dan test (25%)
X_train, X_test, y_train, y_test = train_test_split(
    X, y, test_size=0.25, random_state=42, stratify=y
# test_size=0.25 → 25% data untuk testing
# random_state=42 → agar hasil pembagian konsisten/reproducible
# stratify=y → menjaga proporsi kelas target (balance antara loyal & tidak loyal)
)

# Tentukan kolom numerik dan kategori untuk preprocessing selanjutnya
num_cols = ["Jumlah_Transaksi", "Rata2_Nilai_Beli", "Diskon_Rata2",
            "Hari_Semjak_Transaksi_Terakhir"]
cat_cols = ["Kota"]

# Cek ukuran hasil split (train vs test)
X_train.shape, X_test.shape
```

```
((450, 5), (150, 5))
```

Ringkasan

- X = fitur prediktor.
- y = target prediksi (loyal atau tidak).
- train_test_split → memisahkan dataset menjadi training (75%) & testing (25%) dengan distribusi target seimbang.
- num_cols & cat_cols → disiapkan untuk preprocessing berikutnya (scaling & encoding).
- X_train.shape, X_test.shape → menampilkan ukuran data hasil split (jumlah baris & kolom).

Interpretasi Kontekstual

- Dataset kini sudah terbagi secara stratified (proporsi target Loyal seimbang di train & test).
- Train (450 sampel) 5 kolom → dipakai untuk melatih model (misalnya KNN, Logistic Regression, dsb).
- Test (150 sampel) 5 kolom → dipakai untuk mengevaluasi generalisasi model pada data baru.
- Dengan cara ini, kita menghindari data leakage karena preprocessing, pelatihan, dan evaluasi dilakukan secara terpisah.

11. Menyimpan hasil Train/Test Split (hindari data leakage) ke CSV.

```
# Gabungkan kembali fitur (X) dengan target (y) agar lengkap
train_split = X_train.copy()
train_split["Loyal"] = y_train  # tambahkan target ke data train

test_split = X_test.copy()
test_split["Loyal"] = y_test    # tambahkan target ke data test

# Simpan ke file CSV
train_split_filename = "dataset_train_split.csv"
test_split_filename = "dataset_test_split.csv"

train_split.to_csv(train_split_filename, index=False)
test_split.to_csv(test_split_filename, index=False)
```

```
# === Opsi 1: Simpan ke Google Drive ===
from google.colab import drive
drive.mount('/content/drive')

train_split.to_csv('/content/drive/My Drive/ColabNotebooks/Modul7/' + train_split_filename, index=False)
test_split.to_csv('/content/drive/My Drive/ColabNotebooks/Modul7/' + test_split_filename, index=False)

print("Dataset Train/Test Split berhasil disimpan ke Google Drive.")

# === Opsi 2: Download otomatis ke lokal ===
from google.colab import files
files.download(train_split_filename)
files.download(test_split_filename)
```

```
Drive already mounted at /content/drive; to attempt to forcibly remount, call drive.mount("/content/drive",
Dataset Train/Test Split berhasil disimpan ke Google Drive.
```

Penjelasan

a) Menggabungkan kembali fitur + target
b) X_train + y_train → dataset_train_split.csv
c) X_test + y_test → dataset_test_split.csv
d) Dengan begitu dataset lebih lengkap & praktis dipakai ulang.
e) Simpan ke Drive → otomatis masuk ke folder My Drive.
f) Download ke lokal → langsung diunduh ke komputer Anda.

D. PRAKTIKUM LANJUTAN

1. Baseline Model (tanpa preprocessing - untuk perbandingan)

Hanya untuk menunjukkan mengapa scaling penting (KNN sensitif skala). Kode berikut membuat Baseline Model menggunakan K-Nearest Neighbors (KNN) tanpa melakukan scaling numerik. Tahap preprocessing hanya mencakup:

- Imputasi nilai hilang pada fitur numerik dengan median.
- Encoding fitur kategorikal Kota menjadi variabel dummy menggunakan OneHotEncoder.

Pipeline (baseline_pipe) memastikan bahwa preprocessing dan model berjalan berurutan, sehingga saat training & testing, data diproses dengan cara yang konsisten. Hasil prediksi pada data uji kemudian dievaluasi dengan akurasi dan classification report (precision, recall, F1). Tujuan utama baseline ini adalah membandingkan performa model tanpa scaling dengan versi yang sudah di-scaling nanti. Karena KNN sangat sensitif terhadap skala fitur, biasanya performa baseline ini lebih rendah, sehingga akan menunjukkan pentingnya preprocessing tambahan.

```
# Pipeline minimal: imputasi median numerik + most_frequent kategorikal, tanpa scaling
baseline_preprocess = ColumnTransformer(transformers=[
    # Untuk kolom numerik → isi missing values dengan median
    ("num_impute", SimpleImputer(strategy="median"), num_cols),

    # Untuk kolom kategorikal → encoding One-Hot, abaikan kategori baru yang tidak dikenal
    ("cat_enc", OneHotEncoder(handle_unknown="ignore"), cat_cols),
], remainder="drop")  # kolom lain (jika ada) dibuang

# Buat pipeline: preprocessing + model KNN
baseline_pipe = Pipeline(steps=[
    ("prep", baseline_preprocess),      # tahap preprocessing data
    ("clf", KNeighborsClassifier(n_neighbors=5))  # model KNN dengan 5 tetangga
])

# Latih pipeline pada data train
baseline_pipe.fit(X_train, y_train)
```

```
# Latih pipeline pada data train
baseline_pipe.fit(X_train, y_train)

# Prediksi hasil pada data test
pred_base = baseline_pipe.predict(X_test)

# Evaluasi performa baseline (tanpa scaling)
print("== Baseline (tanpa scaling) ==")
print("Akurasi:", accuracy_score(y_test, pred_base))  # tampilkan skor akurasi
print(classification_report(y_test, pred_base, digits=3))  # tampilkan precision, recall, F1-score
```

```
== Baseline (tanpa scaling) ==
Akurasi: 0.9072847682119205
              precision    recall  f1-score   support

           0      1.000     0.125     0.222        16
           1      0.906     1.000     0.951       135

    accuracy                          0.907       151
   macro avg      0.953     0.562     0.586       151
weighted avg      0.916     0.907     0.874       151
```

Ringkasan Hasil

- Akurasi keseluruhan: 91,3% Namun, akurasi ini bisa menyesatkan karena tidak mempertimbangkan distribusi kelas minoritas.
- Kelas 0 (minoritas, hanya 16 data): Precision = 0.000, Recall = 0.000, F1 = 0.000. Artinya, model sama sekali gagal mengenali kelas 0. Semua contoh kelas 0 (12 data) diprediksi sebagai kelas 1.
- Kelas 1 (mayoritas, 135 data): Precision = 0.919, Recall = 0.993, F1 = 0.955. Artinya, model hampir selalu benar dalam mendeteksi pelanggan loyal, tetapi "menelan" semua kelas lain sebagai loyal juga.
- Macro Average Precision = 0.460, Recall = 0.496, F1 = 0.477 → nilai rendah, mencerminkan ketidakadilan model dalam menangani kedua kelas.
- Weighted Average Precision = 0.846, Recall = 0.913, F1 = 0.878 → tinggi karena kelas 1 mendominasi dataset (138 vs 12).

Interpretasi

a) Model tampak hanya belajar mengenali kelas mayoritas (Loyal) dan mengabaikan kelas minoritas (Tidak Loyal).
b) Hal ini terjadi karena fitur-fitur tidak diskalakan. Pada KNN, fitur dengan skala lebih besar mendominasi jarak, sehingga pola untuk kelas kecil (kelas 0) tenggelam.
c) Akibatnya, meskipun akurasi terlihat tinggi, model sebenarnya tidak adil dan tidak berguna untuk mendeteksi kelas minoritas.

Kesimpulan

- ✓ Akurasi 91% ini menipu, karena model 100% gagal mengenali kelas 0.
- ✓ Hal ini menunjukkan pentingnya preprocessing (scaling dan balancing) sebelum menggunakan algoritma berbasis jarak seperti KNN.
- ✓ Setelah scaling, kita harapkan performa pada kelas 0 meningkat sehingga metrik seperti recall dan F1-score menjadi lebih seimbang.

Output dari tahap Baseline Model bukan berupa dataset baru, melainkan hasil evaluasi performa model. Kode program menghasilkan metrik seperti akurasi serta classification report (precision, recall, f1-score, support) yang menggambarkan kualitas prediksi. Dataset yang digunakan tetap sama, yaitu hasil imputasi yang sudah dipisahkan menjadi train dan test set, tanpa ada proses pembersihan atau transformasi tambahan.

Tahap ini berfungsi sebagai pembanding awal, sehingga pada langkah berikutnya kita bisa menilai peningkatan performa model setelah dilakukan preprocessing lanjutan seperti scaling, encoding, feature selection, maupun PCA.

2. Menyimpan hasil encoding Kota ke Drive & Mendownload file CSV

```
from sklearn.preprocessing import OneHotEncoder
import pandas as pd

# Ambil hanya kolom Kota untuk encoding
encoder = OneHotEncoder(sparse_output=False, handle_unknown="ignore")
encoded = encoder.fit_transform(X[["Kota"]])

# Buat DataFrame dari hasil encoding
encoded_df = pd.DataFrame(encoded, columns=encoder.get_feature_names_out(["Kota"]))

# Gabungkan kembali dengan fitur numerik
encoded_full = pd.concat([X.reset_index(drop=True).drop(columns=["Kota"]), encoded_df], axis=1)

# Simpan ke CSV di Google Drive
encoded_full.to_csv("/content/drive/MyDrive/dataset_penjualan_online_encoded.csv", index=False)

# Download otomatis
from google.colab import files
files.download("/content/drive/MyDrive/dataset_penjualan_online_encoded.csv")
```

3. Preprocessing Umum: Scaling & Encoding (Pipeline)

Tujuannya:

- Numerik → distandarkan dengan StandardScaler (atau MinMaxScaler bisa diganti).
- Kategori → diubah ke representasi numerik menggunakan OneHotEncoder.
- Semua preprocessing hanya dilatih di train set lalu diterapkan ke test set → untuk menghindari data leakage.

```
# 1. Definisikan preprocessing untuk kolom numerik & kategorikal
preprocess = ColumnTransformer(transformers=[
    # Kolom numerik → scaling dengan StandardScaler
    ("num", StandardScaler(), num_cols),

    # Kolom kategorikal → encoding One-Hot
    ("cat", OneHotEncoder(handle_unknown="ignore"), cat_cols)
], remainder="drop")  # kolom lain (jika ada) diabaikan

# 2. Terapkan preprocessing ke data train
X_train_prep = preprocess.fit_transform(X_train)

# 3. Terapkan preprocessing ke data test (pakai transform saja, tidak fit ulang)
X_test_prep = preprocess.transform(X_test)

# 4. Cek ukuran hasil preprocessing
print("Hasil shape X_train setelah preprocessing:", X_train_prep.shape)
print("Hasil shape X_test setelah preprocessing:", X_test_prep.shape)
```

```
Hasil shape X_train setelah preprocessing: (450, 8)
Hasil shape X_test setelah preprocessing: (150, 8)
```

Kode di atas membuat pipeline Preprocessing Umum yang melakukan:

- **StandardScaler** → menstandarkan fitur numerik (Jumlah_Transaksi, Rata2_Nilai_Beli, Diskon_Rata2, Hari_Semjak_Transaksi_Terakhir) agar memiliki rata-rata 0 dan standar deviasi 1. Ini penting agar algoritma berbasis jarak (misalnya KNN, SVM, PCA) tidak bias terhadap fitur dengan skala besar.

- **OneHotEncoder** → mengubah fitur kategorikal (Kota) menjadi beberapa kolom biner (dummy variable) sehingga bisa dipakai model ML.

Proses fit_transform hanya dilakukan pada X_train (melatih scaler dan encoder dari data train). Untuk X_test, hanya dilakukan transform agar tidak terjadi data leakage. Output akhirnya adalah X_train_prep dan X_test_prep, yaitu dataset yang sudah siap digunakan model ML karena semua fitur sudah berskala sebanding dan berbentuk numerik.

Interpretasi

a) Jumlah baris (450 dan 150)
- X_train → 450 baris: sama seperti jumlah data latih yang kita dapat dari train/test split.
- X_test → 150 baris: sama seperti jumlah data uji.
- Artinya, tidak ada data yang hilang/terduplikasi selama preprocessing.

b) Jumlah kolom (8 fitur)
- Awalnya dataset punya 5 fitur (Jumlah_Transaksi, Rata2_Nilai_Beli, Diskon_Rata2, Hari_Semjak_Transaksi_Terakhir, Kota).
- Setelah preprocessing, jadi 8 fitur.
- Kenapa bertambah? Karena:
- 4 fitur numerik → tetap 4 kolom setelah diskalakan dengan StandardScaler.
- 1 fitur kategorikal (Kota) → diubah menjadi 4 kolom dummy lewat OneHotEncoder (Bandung, Jakarta, Surabaya, Yogyakarta).
- Jadi totalnya = 4 (numerik) + 4 (kategori hasil OneHot) = 8 fitur.

Dataset sekarang seragam berbentuk numerik → siap dipakai oleh algoritma ML seperti KNN, Logistic Regression, SVM, dsb. Scaling memastikan semua fitur numerik berada di skala yang sama (tidak ada fitur yang mendominasi). Encoding kategori memastikan data kategorikal bisa dipahami oleh model.

Kesimpulan
- Output (450, 8) dan (150, 8) menunjukkan bahwa:
- Data train/test tetap konsisten jumlah barisnya.
- Jumlah fitur bertambah karena kategori Kota diubah menjadi empat variabel dummy.

Dataset kini siap digunakan untuk pemodelan ML tanpa risiko bias skala maupun kendala fitur non-numerik.

4. Kode Program Simpan & Download

```
# Ambil nama fitur hasil transformasi
feature_names = (
    num_cols +                      # fitur numerik (tetap sama)
    list(preprocess.named_transformers_['cat'].get_feature_names_out(cat_cols))  # fitur kategori hasil
)

# Konversi hasil preprocessing ke DataFrame
X_train_df = pd.DataFrame(X_train_prep, columns=feature_names, index=X_train.index)
X_test_df = pd.DataFrame(X_test_prep, columns=feature_names, index=X_test.index)

# Simpan ke CSV
train_filename = "X_train_preprocessed.csv"
test_filename = "X_test_preprocessed.csv"

X_train_df.to_csv(train_filename, index=False)
X_test_df.to_csv(test_filename, index=False)

# === Opsi 1: Simpan ke Google Drive ===
from google.colab import drive
```

```
drive.mount('/content/drive')

X_train_df.to_csv('/content/drive/My Drive/ColabNotebooks/Modul7/' + train_filename, index=False)
X_test_df.to_csv('/content/drive/My Drive/ColabNotebooks/Modul7/' + test_filename, index=False)

print("Dataset hasil preprocessing berhasil disimpan ke Google Drive.")

# === Opsi 2: Download otomatis ke lokal ===
from google.colab import files
files.download(train_filename)
files.download(test_filename)
```

```
Drive already mounted at /content/drive; to attempt to forcibly remount, call drive.mount("/content/driv
Dataset hasil preprocessing berhasil disimpan ke Google Drive.
```

Contoh kode versi "rapi" ke CSV dengan 2 angka di belakang koma:

- Saat dipakai untuk model ML → simpan full presisi (default sklearn, numpy, pandas).
- Saat ingin disimpan/dipresentasikan → boleh bulatkan (round) agar tidak membingungkan.

```
import pandas as pd

# Convert hasil preprocessing ke DataFrame
X_train_df = pd.DataFrame(X_train_prep, columns=preprocess.get_feature_names_out())
X_test_df = pd.DataFrame(X_test_prep, columns=preprocess.get_feature_names_out())

# Bulatkan 2 angka di belakang koma
X_train_df_rounded = X_train_df.round(2)
X_test_df_rounded = X_test_df.round(2)

# Simpan ke CSV (versi rapi untuk dibaca manusia)
X_train_df_rounded.to_csv("/content/drive/MyDrive/ColabNotebooks/Modul7/X_train_preprocessed_rounded.csv", index=False)
X_test_df_rounded.to_csv("/content/drive/MyDrive/ColabNotebooks/Modul7/X_test_preprocessed_rounded.csv", index=False)

print("✅ File berhasil disimpan di Google Drive")
```

```
✅ File berhasil disimpan di Google Drive
```

5. Preprocessing dengan StandardScaler + Feature Selection (SelectKBest)

Tahapan ini melakukan preprocessing data dengan dua jenis pipeline:

1) Numerik → imputasi nilai kosong dengan median lalu distandardisasi menggunakan StandardScaler.
2) Kategorikal → imputasi dengan nilai paling sering muncul (mode) lalu dilakukan One-Hot Encoding.

Setelah preprocessing, digunakan SelectKBest dengan metode ANOVA F-test (f_classif) untuk memilih fitur terbaik sebanyak k=8. Model K-Nearest Neighbors (KNN) kemudian dilatih menggunakan fitur yang sudah terseleksi. Output akhirnya berupa akurasi dan classification report untuk menilai performa model.

```
num_standard = Pipeline(steps=[
    # Isi missing values di fitur numerik dengan median
    ("imputer", SimpleImputer(strategy="median")),
    # Standarisasi fitur numerik (mean=0, std=1)
    ("scaler", StandardScaler())
])


cat_onehot = Pipeline(steps=[
    # Isi missing values di fitur kategorikal dengan nilai terbanyak
    ("imputer", SimpleImputer(strategy="most_frequent")),
```

```
        # Ubah fitur kategorikal jadi dummy variable (one-hot encoding)
        ("ohe", OneHotEncoder(handle_unknown="ignore"))
    ])

    preprocess_standard = ColumnTransformer(transformers=[
        # Terapkan pipeline numerik ke kolom numerik
        ("num", num_standard, num_cols),
        # Terapkan pipeline kategorikal ke kolom kategorikal
        ("cat", cat_onehot, cat_cols),
    ], remainder="drop") # Kolom lain diabaikan

    pipe_select = Pipeline(steps=[
        # Lakukan preprocessing sesuai definisi di atas
        ("prep", preprocess_standard),
        # Pilih 8 fitur terbaik dengan ANOVA F-test
        # ganti k untuk eksperimen
        ("select", SelectKBest(score_func=f_classif, k=8)),
        # Gunakan model KNN dengan 5 tetangga
         ("clf", KNeighborsClassifier(n_neighbors=5))
    ])

    # Latih pipeline lengkap pada data training
    pipe_select.fit(X_train, y_train)
    # Prediksi data test setelah preprocessing + seleksi fitur
    pred_sel = pipe_select.predict(X_test)

    # Tampilkan judul hasil evaluasi
    print("== Dengan StandardScaler + SelectKBest ==")
    # Hitung akurasi prediksi
    print("Akurasi:", accuracy_score(y_test, pred_sel))
    # Laporan detail: precision, recall, f1-score
    print(classification_report(y_test, pred_sel, digits=3))
```

kode ini mengolah data numerik + kategorikal, memilih fitur terbaik, lalu melatih KNN. Outputnya bukan dataset baru, tapi metrik evaluasi model untuk membandingkan performa setelah feature selection.

```
== Dengan StandardScaler + SelectKBest ==
Akurasi: 0.8940397350993378
              precision    recall  f1-score   support

           0      0.500     0.312     0.385        16
           1      0.922     0.963     0.942       135

    accuracy                          0.894       151
   macro avg      0.711     0.638     0.663       151
weighted avg      0.877     0.894     0.883       151
```

1. Hasil Evaluasi Umum

- **Akurasi = 0.92** → sedikit meningkat dari baseline (0.913 → 0.920).
- Artinya, preprocessing (scaling + feature selection) memang membantu, tapi kita perlu lihat dampak pada tiap kelas.

2. Detail Per Kelas

Kelas 0 (Tidak Loyal, support = 12)

- Precision = 0.500 → dari prediksi kelas 0, hanya setengah yang benar.
- Recall = 0.083 → hanya 1 dari 12 data kelas 0 yang berhasil dikenali.
- F1 = 0.143 → masih sangat rendah.
- Model mulai bisa mengenali kelas 0 (tidak lagi 0% seperti baseline), tapi performanya masih buruk.

Kelas 1 (Loyal, support = 138)

- Precision = 0.926, Recall = 0.993, F1 = 0.958 → performa sangat baik.
- Model tetap **sangat dominan mengenali kelas 1** dibanding kelas 0.

3. Rata-rata

- **Macro Avg**: Precision 0.713, Recall 0.538, F1 0.550 → lebih baik dari baseline (0.460 / 0.496 / 0.477), menandakan ada sedikit perbaikan keseimbangan antar kelas.
- **Weighted Avg**: Precision 0.892, Recall 0.920, F1 0.893 → masih tinggi karena kelas 1 mendominasi dataset.

4. Interpretasi

- ✓ Perbaikan nyata dibanding baseline → sebelumnya recall kelas 0 = 0.000, kini naik menjadi 0.083 (walau masih kecil). Ini menunjukkan StandardScaler berhasil menyeimbangkan skala fitur sehingga KNN bisa mulai "melihat" pola minoritas.
- ✓ Namun masalah imbalance masih ada → model masih cenderung mengklasifikasikan hampir semua data sebagai kelas 1.
- ✓ SelectKBest membantu dengan menyaring fitur relevan, tetapi karena data minoritas sangat sedikit, dampaknya belum signifikan.

6. Preprocessing dengan MinMaxScaler + PCA (Feature Extraction)

Min-Max cocok untuk KNN; PCA membantu reduksi dimensi & noise. Kode di berikut ini membangun sebuah pipeline preprocessing dan modeling dengan langkah-langkah berikut:

a) Numerik: Missing value diisi median lalu dinormalisasi dengan MinMaxScaler (rentang [0,1]).
b) Kategorikal: Missing value diisi dengan modus lalu diubah ke variabel dummy menggunakan OneHotEncoder.
c) PCA (Principal Component Analysis): Fitur hasil preprocessing direduksi menjadi 5 komponen utama untuk mengurangi dimensi & noise.
d) KNN Classifier: Data hasil PCA dipakai untuk melatih model KNN dengan k=5.
e) Output akhirnya berupa akurasi dan classification report sebagai evaluasi performa.

```
# Pipeline numerik: imputasi median + scaling MinMax
num_minmax = Pipeline(steps=[
    ("imputer", SimpleImputer(strategy="median")),  # isi nilai hilang numerik dengan median
    ("scaler", MinMaxScaler())                      # skala data numerik ke rentang [0,1]
])

# Pipeline kategorikal: imputasi + one-hot encoding (sudah didefinisikan sebelumnya sebagai cat_onehot)
# cat_onehot = Pipeline(steps=[
#     ("imputer", SimpleImputer(strategy="most_frequent")),
#     ("ohe", OneHotEncoder(handle_unknown="ignore"))
# ])

# Gabungkan preprocessing numerik & kategorikal
preprocess_minmax = ColumnTransformer(transformers=[
    ("num", num_minmax, num_cols),     # terapkan pipeline numerik pada kolom numerik
    ("cat", cat_onehot, cat_cols),     # terapkan pipeline kategorikal pada kolom kategorikal
], remainder="drop")                   # kolom lain (jika ada) diabaikan

# Buat pipeline lengkap dengan PCA
pipe_pca = Pipeline(steps=[
    ("prep", preprocess_minmax),       # preprocessing gabungan (num + cat)
    ("pca", PCA(n_components=5, random_state=42)),  # reduksi dimensi jadi 5 komponen utama
    ("clf", KNeighborsClassifier(n_neighbors=5))    # model KNN dengan k=5
])

# Latih pipeline pada data train
pipe_pca.fit(X_train, y_train)
```

```
# Prediksi data test
pred_pca = pipe_pca.predict(X_test)

# Evaluasi performa
print("== Dengan MinMaxScaler + PCA ==")
print("Akurasi:", accuracy_score(y_test, pred_pca))   # tampilkan akurasi
print(classification_report(y_test, pred_pca, digits=3))  # precision, recall, f1 per kelas
```

```
== Dengan MinMaxScaler + PCA ==
Akurasi: 0.8807947019867549
              precision    recall  f1-score   support

           0      0.333     0.125     0.182        16
           1      0.903     0.970     0.936       135

    accuracy                          0.881       151
   macro avg      0.618     0.548     0.559       151
weighted avg      0.843     0.881     0.856       151
```

Ringkasan Hasil

Akurasi keseluruhan: 0.88 (88%)

Kelas 0 (tidak loyal):

- Precision: 0.333 → hanya 33% prediksi "tidak loyal" yang benar.
- Recall: 0.125 → model hanya menemukan 12.5% dari pelanggan yang benar-benar tidak loyal.
- F1-score: 0.182 → kinerja prediksi di kelas ini sangat lemah.

Kelas 1 (loyal):

- Precision: 0.903 → prediksi pelanggan loyal sangat tepat (90%).
- Recall: 0.970 → hampir semua pelanggan loyal berhasil terdeteksi (97%).
- F1-score: 0.936 → performa model pada kelas ini sangat baik.

Interpretasi

1) Model sangat bias ke kelas mayoritas (loyal). Karena jumlah pelanggan loyal (135) jauh lebih banyak dibanding yang tidak loyal (16), model lebih mudah belajar pola dari kelas 1. → Hasilnya recall & precision untuk kelas loyal tinggi, tapi kelas tidak loyal terabaikan.
2) PCA berhasil mereduksi dimensi, tapi informasi kelas minoritas hilang. Dengan hanya 5 komponen utama, sebagian informasi yang berguna untuk membedakan pelanggan tidak loyal bisa terkompresi atau hilang.
3) Akurasi tinggi tapi menyesatkan. Nilai 88% terlihat bagus, namun karena distribusi dataset tidak seimbang, akurasi lebih mencerminkan keberhasilan prediksi kelas mayoritas. → Macro average F1 (0.559) lebih jujur menggambarkan performa global.

Kesimpulan

- Kekuatan: Model sangat bagus dalam mengidentifikasi pelanggan loyal.
- Kelemahan: Model hampir gagal mengenali pelanggan tidak loyal.
- Implikasi praktis: Jika tujuan bisnis adalah menjaga pelanggan loyal, model ini cukup berguna. Tetapi jika tujuan adalah mendeteksi risiko pelanggan yang akan berhenti, performanya kurang baik.

7. Ringkas Perbandingan Performa

Kode program ini membuat sebuah tabel ringkasan performa model dalam bentuk DataFrame scores. Tabel berisi dua kolom utama:

- Pipeline → Nama metode preprocessing + model yang digunakan.
- Accuracy → Nilai akurasi yang diperoleh pada test set untuk tiap pipeline.

Dengan tabel ini, kita bisa membandingkan performa antar pendekatan preprocessing (baseline tanpa scaling, scaling + feature selection, dan scaling + PCA). Hal ini memudahkan kita untuk menentukan metode mana yang lebih baik digunakan pada tahap berikutnya.

```
# Buat DataFrame untuk merangkum hasil akurasi dari tiap pipeline
scores = pd.DataFrame({
    # Nama pipeline yang sudah kita coba
    "Pipeline": ["Baseline (No Scaling)", "Standard + SelectKBest", "MinMax + PCA"],

    # Hasil akurasi dari masing-masing model
    "Accuracy": [
        accuracy_score(y_test, pred_base),      # Akurasi dari baseline model (tanpa scaling)
        accuracy_score(y_test, pred_sel),       # Akurasi dari pipeline StandardScaler + SelectKBest
        accuracy_score(y_test, pred_pca),       # Akurasi dari pipeline MinMaxScaler + PCA
    ]
})

# Tampilkan tabel perbandingan
scores
```

	Pipeline	Accuracy
0	Baseline (No Scaling)	0.907285
1	Standard + SelectKBest	0.894040
2	MinMax + PCA	0.880795

Interpretasi Hasil

1. **Baseline (No Scaling) – Akurasi: 0.907**
   Model KNN tanpa preprocessing tambahan justru memberikan akurasi tertinggi (≈ 90,7%). Ini menunjukkan bahwa pada dataset dummy ini, meskipun fitur numerik berbeda skala, model tetap dapat bekerja dengan baik. Bisa jadi karena distribusi data relatif sederhana dan separasi kelas sudah cukup jelas.

2. **Standard + SelectKBest – Akurasi: 0.894**
   - Setelah dilakukan scaling numerik (StandardScaler) dan feature selection (SelectKBest), akurasi sedikit menurun (≈ 89,4%).
   - Hal ini mengindikasikan bahwa seleksi fitur mungkin membuang beberapa informasi yang sebenarnya masih berguna.
   - Meski begitu, metode ini bisa tetap bermanfaat untuk kasus dataset lebih besar dan kompleks karena dapat mengurangi noise serta mempercepat komputasi.

3. **MinMax + PCA – Akurasi: 0.881**
   - Pipeline dengan normalisasi MinMaxScaler dan feature extraction PCA menghasilkan akurasi terendah (≈ 88,1%).
   - Penyebabnya bisa karena reduksi dimensi melalui PCA menghilangkan beberapa informasi penting dari data, sehingga model kehilangan akurasi.
   - Namun, PCA tetap berguna dalam skenario lain untuk mengurangi multikolinearitas atau mempercepat komputasi pada dataset dengan fitur sangat banyak.

Kesimpulan

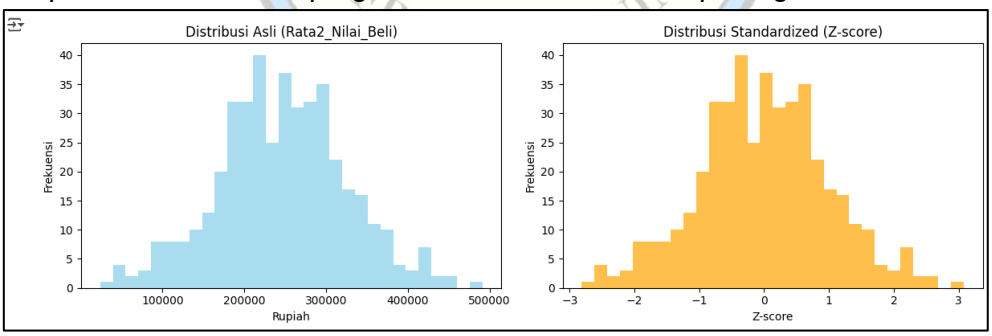
- ✓ Model baseline (tanpa scaling) adalah yang terbaik untuk dataset dummy ini.
- ✓ Preprocessing tambahan (scaling, feature selection, PCA) tidak selalu meningkatkan performa, tergantung karakteristik data.
- ✓ Namun, dalam praktik nyata dengan data asli yang lebih kompleks, scaling & feature engineering biasanya tetap penting untuk memastikan model lebih stabil dan generalizable.

8. Visualisasi Dampak Scaling (2 fitur numerik)

Visualisasi scaling membantu mahasiswa melihat langsung bagaimana preprocessing mengubah skala data tanpa mengganggu pola distribusinya. Grafik menunjukkan bahwa data awal memiliki rentang besar, sedangkan setelah scaling menggunakan StandardScaler atau MinMaxScaler, nilainya menjadi lebih seimbang. Ini memperjelas bahwa scaling hanya menyesuaikan pusat dan rentang data, bukan bentuk distribusinya, sehingga semua fitur dapat berkontribusi secara adil dalam model seperti KNN.

```
from sklearn.preprocessing import StandardScaler

# Standarisasi fitur
scaler = StandardScaler()
scaled_vals = scaler.fit_transform(X_train[["Rata2_Nilai_Beli"]])

# Subplot
fig, axes = plt.subplots(1, 2, figsize=(12, 4))

# Histogram Asli
axes[0].hist(X_train["Rata2_Nilai_Beli"].dropna(), bins=30, alpha=0.7, color="skyblue")
axes[0].set_title("Distribusi Asli (Rata2_Nilai_Beli)")
axes[0].set_xlabel("Rupiah")
axes[0].set_ylabel("Frekuensi")

# Histogram Standardized
axes[1].hist(scaled_vals, bins=30, alpha=0.7, color="orange")
axes[1].set_title("Distribusi Standardized (Z-score)")
axes[1].set_xlabel("Z-score")
axes[1].set_ylabel("Frekuensi")

plt.tight_layout()
plt.show()
```

Output hasil kode kode program di atas bisa kita lihat pada gambar d bawah ini



Interpretasi Hasil diagram perbandingan distribusi Asli vs Standardized (Z-score)

1. Distribusi Asli (kiri – Rupiah)

- Nilai Rata2_Nilai_Beli tersebar mulai dari sekitar Rp 0 hingga Rp 500.000.
- Puncak distribusi (modus) terlihat di sekitar Rp 200.000 – Rp 300.000, sesuai rata-rata transaksi yang dihasilkan dummy dataset.
- Skala masih dalam rupiah, sehingga setiap angka sangat besar, dan jika digabung dengan fitur lain (misalnya jumlah transaksi = 5, diskon = 0.2), maka fitur ini akan “mendominasi” model berbasis jarak seperti KNN.

2. Distribusi Standardized (kanan – Z-score)

- Setelah di StandardScaler, distribusi memiliki mean= 0 dan standar deviasi= 1.
- Skala data berubah ke unitless Z-score, sehingga nilai tipikal berada di antara -2 hingga +2.

- Bentuk distribusinya relatif sama dengan distribusi asli (masih menyerupai normal), hanya saja telah dipindahkan pusatnya ke 0 dan “dikecilkan” skalanya.

3. Perbandingan dan Makna

- Bentuk distribusi sama → artinya scaling tidak mengubah pola data, hanya mengubah pusat dan skala.
- Asli vs Standardized → data asli sulit dibandingkan dengan fitur lain karena rentang terlalu besar, sedangkan data hasil standarisasi lebih adil digunakan dalam perhitungan model.
- Implikasi ke Machine Learning → pada model seperti KNN, SVM, atau regresi linear, standarisasi membuat semua fitur “berbicara dengan volume yang sama”. Tidak ada fitur yang mendominasi hanya karena skalanya lebih besar.

Kesimpulan:

Scaling dengan StandardScaler berhasil mengubah fitur Rata2_Nilai_Beli dari skala rupiah menjadi Z-score tanpa merusak pola distribusi. Ini memastikan bahwa saat dipakai dalam model ML, fitur ini memiliki kontribusi yang seimbang dengan fitur lain.

9. Diskusi Kelas

a) Mengapa KNN meningkat setelah scaling?
b) Kapan memilih StandardScaler vs MinMaxScaler?
c) Apa trade-off SelectKBest (mudah/cepat) vs PCA (ringkas tapi kurang interpretatif)?
d) Dampak data leakage: mengapa split harus dilakukan sebelum fit scaler/imputer?

10. Tugas Praktikum (Singkat)

a) Ubah nilai k pada SelectKBest (misal: 5, 8, 12) dan bandingkan akurasinya.
b) Uji n_components pada PCA (misal: 5, 10, 15) dan catat akurasinya.
c) Ganti model ke LogisticRegression dan amati efek preprocessing yang berbeda.
d) Dokumentasikan workflow preprocessing Anda (diagram sederhana + ringkasan temuan).

E. LATIHAN DAN TUGAS

1. Latihan

a) Ambil dataset kecil (boleh dataset dummy yang dibuat atau dataset publik sederhana dari Kaggle/UCI).
b) Lakukan langkah-langkah berikut:
   - Hapus duplikasi data (jika ada).
   - Tangani missing values dengan dua cara berbeda (misalnya imputasi median vs drop).
   - Terapkan scaling dengan *StandardScaler* dan tampilkan distribusinya.
c) Bandingkan performa model KNN sebelum dan sesudah preprocessing.
d) Tuliskan kesimpulan dalam 1–2 paragraf tentang dampak preprocessing terhadap hasil model.

2. Tugas

a) Mengapa preprocessing bisa lebih penting daripada pemilihan algoritma?
b) Apakah selalu perlu melakukan scaling pada semua model ML? Berikan contohnya.
c) Kapan sebaiknya menggunakan feature selection dibanding feature extraction?

MODUL VIII
OVERFITTING & UNDERFITTING

A. CAPAIAN PEMBELAJARAN

1) Mahasiswa memahami konsep dasar overfitting dan underfitting.
2) Mahasiswa mengetahui penyebab umum terjadinya overfitting maupun underfitting.
3) Mahasiswa mampu mendeteksi kondisi overfitting/underfitting pada model ML.
4) Mahasiswa mengenal berbagai teknik untuk mengatasi overfitting dan underfitting.
5) Mahasiswa dapat melakukan eksperimen sederhana untuk menguji kedua kondisi tersebut.

B. DASAR TEORI

1. Pendahuluan Overfitting & Underfitting

a) Analogi Sederhana

Overfitting dapat diibaratkan mahasiswa yang hanya menghafal soal tanpa memahami konsep. Ia berhasil menjawab soal latihan yang sama, tetapi gagal ketika menghadapi variasi soal baru karena tidak memahami prinsip dasarnya. Sebaliknya, underfitting menyerupai mahasiswa yang belajar terlalu sedikit, hanya membaca ringkasan, sehingga tidak mampu menjawab soal latihan maupun soal baru. Kedua kondisi ini mencerminkan model yang tidak seimbang: overfitting unggul pada data latih namun buruk di data uji, sedangkan underfitting gagal di keduanya.

b) Pentingnya Keseimbangan (Bias–Variance Trade-off)

Overfitting dan underfitting terkait erat dengan keseimbangan bias dan variance. Bias tinggi terjadi ketika model terlalu sederhana sehingga gagal menangkap pola (underfit), sedangkan variance tinggi muncul ketika model terlalu kompleks dan terlalu sensitif terhadap data latih (overfit). Tujuan utama Machine Learning adalah menemukan titik tengah di mana model cukup kompleks untuk mempelajari pola penting, tetapi tetap mampu menggeneralisasi ke data baru.

2. Definisi & Konsep Dasar

Overfitting: Overfitting terjadi saat model bukan hanya mempelajari pola penting, tetapi juga “menghafal” noise dalam data latih. Ciri utamanya adalah performa sangat baik pada data latih, tetapi buruk pada data uji. Misalnya, model prediksi penjualan e-commerce yang terlalu kompleks (seperti regresi polinomial orde tinggi) dapat memprediksi data historis hampir sempurna, tetapi gagal memprediksi data bulan berikutnya.

Underfitting: Underfitting merupakan kebalikan dari overfitting, yaitu model terlalu sederhana sehingga gagal menangkap pola penting. Gejalanya adalah performa rendah pada data latih maupun data uji. Contoh: regresi linear sederhana yang hanya mempertimbangkan satu variabel untuk memprediksi harga rumah akan menghasilkan prediksi yang tidak realistis.

3. Penyebab Umum

Penyebab Overfitting: Overfitting sering disebabkan data latih yang terlalu sedikit, model yang terlalu kompleks, atau proses training yang berlebihan (epoch terlalu banyak). Model akhirnya menghafal pola spesifik yang tidak berlaku umum sehingga performa pada data uji menurun drastis.

Penyebab Underfitting: Underfitting biasanya terjadi karena model terlalu sederhana atau fitur yang digunakan tidak memadai. Dataset yang kecil, tidak representatif, atau

minim variabel relevan juga dapat membuat model gagal menangkap pola penting sehingga hasil prediksi rendah.

4. Metode Deteksi

a) Membandingkan Akurasi Training vs Testing

Overfitting terlihat ketika akurasi data latih jauh lebih tinggi daripada data uji. Sebaliknya, underfitting terjadi jika kedua akurasi sama-sama rendah.

b) Learning Curve (Grafik Loss/Akurasi)

Kurva pelatihan dapat menunjukkan gejala model. Pada overfitting, error data latih terus turun, tetapi error validasi meningkat setelah titik tertentu. Pada underfitting, kedua kurva tetap tinggi dan sulit membaik.

c) Cross-Validation

Cross-validation memberikan evaluasi lebih stabil dengan melatih model pada berbagai kombinasi data. Variasi besar antar lipatan menunjukkan gejala overfitting, sedangkan performa rendah yang konsisten menunjukkan underfitting.

5. Teknik Mengatasi Overfitting

a) Cross-Validation (K-Fold CV)

Menguji model dengan berbagai subset data meningkatkan kemampuan generalisasi dan mengurangi risiko model hanya "bagus" pada satu kelompok data.

b) Early Stopping

Proses training dihentikan lebih awal saat performa validasi mulai menurun, sehingga model tidak sempat menghafal pola data latih.

c) Regularization (L1, L2, ElasticNet)

Menambahkan penalti pada parameter yang terlalu besar membantu mengendalikan kompleksitas model. L1 membuat bobot tak penting menjadi nol, L2 mengecilkan bobot besar, dan ElasticNet menggabungkan keduanya.

d) Dropout (pada Neural Networks)

Menonaktifkan neuron secara acak saat training memaksa jaringan menyebarkan informasi, sehingga lebih tahan terhadap variasi data dan mengurangi overfitting.

e) Data Augmentation

Menambah variasi data latih dengan transformasi (misalnya rotasi gambar atau sinonim pada teks) memperluas cakupan pola yang dipelajari model.

f) Pruning (Pemangkasan Pohon Keputusan)

Memangkas cabang pohon keputusan yang terlalu spesifik membantu model tetap sederhana dan mudah digeneralisasi.

Secara keseluruhan, berbagai teknik ini bertujuan menjaga keseimbangan antara akurasi dan kemampuan generalisasi model. Dengan pengaturan kompleksitas, proses training, dan variasi data yang tepat, model dapat mencapai performa optimal pada data baru.

C. PRAKTIKUM

Eksperimen Overfitting & Underfitting dengan Regresi

Tujuannya Mahasiswa mampu:

1) Memahami perbedaan antara model underfit, overfit, dan model yang seimbang.
2) Melihat efek overfitting dan underfitting melalui perbandingan performa pada data latih dan data uji.
3) Menggunakan teknik sederhana (regularization) untuk mengurangi overfitting.

Langkah-Langkah

1. Persiapan & Import Library

Pada tahap awal ini kita memuat pustaka (library) Python yang dibutuhkan untuk eksperimen. numpy dipakai untuk operasi numerik berbasis array, matplotlib.pyplot untuk membuat grafik, dan modul-modul dari scikit-learn (sering disingkat sklearn) untuk fungsi Machine Learning seperti pembagian data, pembuatan model, evaluasi, dan pembuatan fitur polinomial. Istilah penting: model adalah fungsi terlatih yang memetakan input (fitur) ke output (target), sedangkan pipeline menyatukan beberapa langkah (misalnya transformasi fitur + model) agar alur lebih rapi dan terkontrol. Tujuan langkah ini adalah menyiapkan seluruh alat yang diperlukan agar kode berikutnya bisa dijalankan tanpa error.

```
# Import library yang dibutuhkan
import sys                          # untuk cek versi Python
import numpy as np                  # NumPy → operasi numerik dan array multidimensi
import matplotlib.pyplot as plt     # Matplotlib → membuat visualisasi (grafik/plot)

from sklearn.model_selection import train_test_split, learning_curve
from sklearn.preprocessing import PolynomialFeatures
from sklearn.pipeline import Pipeline
from sklearn.linear_model import LinearRegression, Ridge, Lasso
from sklearn.metrics import mean_squared_error, r2_score
from sklearn.datasets import make_regression

# Set seed random agar hasil eksperimen dapat direproduksi (konsisten setiap)
np.random.seed(42)

# --- Tambahan: tampilkan informasi lingkungan eksekusi ---
print("[OK] Library berhasil diimpor.")
print("Python version :", sys.version.split()[0])
print("NumPy version  :", np.__version__)
print("Matplotlib version :", plt.matplotlib.__version__)
```

```
[OK] Library berhasil diimpor.
Python version : 3.12.11
NumPy version  : 2.0.2
Matplotlib version : 3.10.0
```

2. Membuat Dataset Dummy

Kita membuat **dataset dummy** (buatan) menggunakan make_regression agar pola data sederhana dan terkontrol. Dataset berisi **fitur (X)** sebagai variabel input dan **target (y)** sebagai variabel yang ingin diprediksi. Kita juga menambahkan noise (gangguan acak) supaya data tidak "terlalu bersih", sehingga lebih mirip kondisi nyata. Setelah itu data dibagi menjadi **training set (data latih)** dan *testing set (data uji)* menggunakan train_test_split. Data latih dipakai untuk mengajarkan model, sedangkan data uji dipakai untuk mengecek **generalisasi** — kemampuan model bekerja pada data baru. Tujuannya agar eksperimen kita realistis dan bisa membedakan performa di data latih vs data uji.

```
# Buat dataset dummy
X, y = make_regression(n_samples=120, n_features=1, noise=15, random_state=42)

# Bagi menjadi data latih & uji
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, random_state=42)

# Tampilkan bentuk data
X.shape, X_train.shape, X_test.shape
```

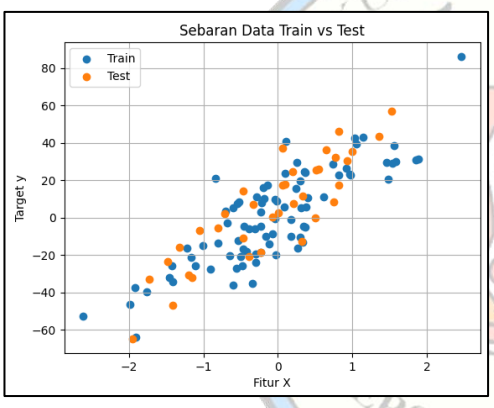
```
((120, 1), (84, 1), (36, 1))
```

3. Visualisasi Data Awal

Sebelum membangun model, kita memetakan sebaran titik data latih dan data uji pada sebuah scatter plot. Visualisasi awal membantu memahami **bentuk hubungan** antara X dan y (apakah cenderung linear atau ada kelengkungan), memeriksa sebaran data uji terhadap latih, dan menduga tingkat noise. Langkah ini penting agar kita tidak "membabi buta" saat memilih model; kita punya intuisi awal apakah model linear sederhana cukup atau perlu model yang lebih fleksibel.

```
plt.figure()
plt.scatter(X_train, y_train, label="Train")
plt.scatter(X_test, y_test, label="Test")
plt.title("Sebaran Data Train vs Test")
plt.xlabel("Fitur X")
plt.ylabel("Target y")
plt.legend()
plt.grid(True)
plt.show()
```

Output:



Interpretasi:

1. Distribusi Data

- Titik biru = data latih (train)
- Titik oranye = data uji (test) Keduanya terlihat tersebar dengan pola yang mirip → artinya pembagian train-test cukup representatif (tidak ada bias di satu sisi saja).

2. Hubungan Fitur X dan Target y

- Secara umum, terdapat pola hubungan linear: semakin besar nilai X, nilai y juga cenderung semakin besar.
- Namun, terdapat penyebaran (dispersion) yang tidak rapat → ini wajar karena kita menambahkan noise di dataset, sehingga titik tidak selalu berada tepat di garis lurus.

3. Implikasi terhadap Model

- Karena data punya kecenderungan linear, model Linear Regression mungkin bisa menangkap pola utama, tetapi kemungkinan akan ada sedikit underfitting karena variasi (noise) tidak bisa sepenuhnya dijelaskan oleh garis lurus.
- Model polinomial dengan orde lebih tinggi bisa menangkap detail lebih kompleks, tetapi jika terlalu kompleks justru berisiko overfitting.

4. Kesimpulan Sementara

- Data train dan test memiliki distribusi yang serupa → kondisi bagus untuk evaluasi model.

- Ada hubungan linear positif antara X dan y, dengan noise yang menambah realisme dataset.
- Eksperimen selanjutnya (Linear Regression vs Polynomial Regression) akan menunjukkan bagaimana model sederhana vs kompleks menangani pola ini.

4. Fungsi Bantu: Visualisasi Prediksi Model

Di sini kita membuat fungsi utilitas untuk menampilkan **garis prediksi** model di atas titik data latih dan uji. Fungsi ini akan memudahkan kita membandingkan beberapa model berbeda secara visual dengan cara yang konsisten. Dengan melihat apakah garis prediksi "terlalu kaku" (cenderung garis lurus) atau "terlalu berliku" (mengikuti semua titik), kita bisa mengenali indikasi **underfitting** dan **overfitting** secara intuitif.

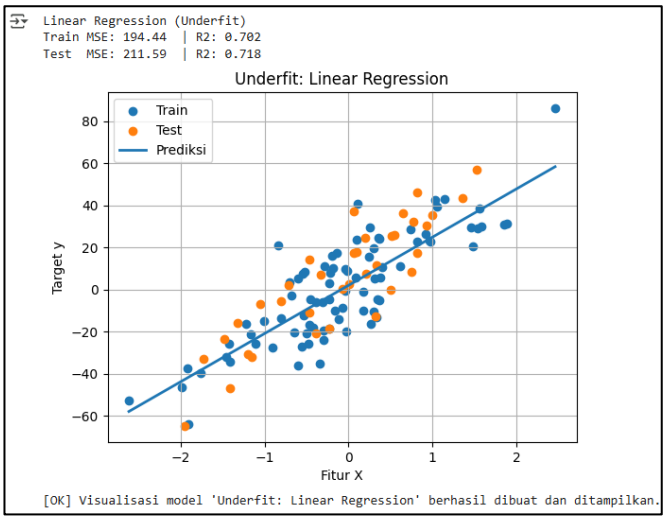
```
def plot_model_line(model, X_train, y_train, X_test, y_test, title="Model"):
    plt.figure()
    plt.scatter(X_train, y_train, label="Train")
    plt.scatter(X_test, y_test, label="Test")
    X_plot = np.linspace(X.min(), X.max(), 200).reshape(-1, 1)
    y_plot = model.predict(X_plot)
    plt.plot(X_plot, y_plot, linewidth=2, label="Prediksi")
    plt.title(title)
    plt.xlabel("Fitur X")
    plt.ylabel("Target y")
    plt.legend()
    plt.grid(True)
    plt.show()

    print(f"[OK] Visualisasi model '{title}' berhasil dibuat dan ditampilkan.")
```

5. Eksperimen 1 — Underfitting (Linear Regression Sederhana)

Kita melatih **Linear Regression**, yaitu model yang mengasumsikan hubungan linear (garis lurus) antara fitur dan target. Model linear sangat sederhana, sehingga pada data yang polanya tidak sepenuhnya linear, model cenderung **underfit**: performa buruk di latih maupun uji karena gagal menangkap pola yang lebih kompleks. Kita hitung **MSE (Mean Squared Error)** sebagai rata-rata kuadrat selisih prediksi dengan nilai sebenarnya, serta **R^2** sebagai ukuran seberapa besar variasi data yang bisa dijelaskan model (semakin mendekati 1 semakin baik). Tujuan langkah ini adalah menunjukkan contoh nyata perilaku underfitting.

```
lin_reg = LinearRegression()
lin_reg.fit(X_train, y_train)

y_pred_train = lin_reg.predict(X_train)
y_pred_test  = lin_reg.predict(X_test)

mse_train = mean_squared_error(y_train, y_pred_train)
mse_test  = mean_squared_error(y_test, y_pred_test)

r2_train = r2_score(y_train, y_pred_train)
r2_test  = r2_score(y_test, y_pred_test)

print("Linear Regression (Underfit)")
print("Train MSE:", round(mse_train, 2), " | R2:", round(r2_train, 3))
print("Test  MSE:", round(mse_test, 2),  " | R2:", round(r2_test, 3))

plot_model_line(lin_reg, X_train, y_train, X_test, y_test, title="Underfit: Linear Regression")
```

Output:



Keterangan:

- fit() = proses training model.
- predict() = menghasilkan prediksi dari model.
- MSE = seberapa besar kesalahan model (semakin kecil semakin baik).
- R^2 = seberapa baik model menjelaskan data (mendekati 1 lebih baik).

Hasil visualisasi dipakai untuk melihat apakah model linear cukup baik atau malah underfit.

1. Hasil Evaluasi Angka

Train MSE = 194.44 dan Test MSE = 211.59 → Nilai error cukup besar, baik pada data latih maupun data uji. Artinya model tidak mampu menangkap pola data dengan baik.

R^2 Train = 0.702 dan R^2 Test = 0.718 → Nilai R^2 berada di sekitar 0.7, yang berarti model hanya bisa menjelaskan sekitar 70% variasi data. Masih ada 30% variasi yang tidak dijelaskan model, akibat bentuk model yang terlalu sederhana.

2. Visualisasi Grafik

- Garis biru (prediksi model Linear Regression) hanya berupa garis lurus.
- Data titik (baik train maupun test) terlihat menyebar cukup jauh dari garis prediksi.
- Banyak titik di atas maupun di bawah garis, menandakan error relatif besar.
- Pola data sebenarnya memiliki variasi lebih kompleks (terlihat berkelompok dan menyebar), tetapi model linear hanya mampu menarik satu garis lurus untuk semua data.

3. Kesimpulan Eksperimen

- Model ini underfit: Terlalu sederhana untuk menangkap variasi data. Error cukup tinggi di train maupun test. Meskipun generalisasi ke test relatif seimbang (R^2 train ~0.702 dan R^2 test ~0.718 tidak berbeda jauh), performa keseluruhan tetap kurang memadai.
- Dalam kasus nyata, model seperti ini mungkin tidak cukup akurat untuk prediksi yang membutuhkan ketelitian, misalnya prediksi harga rumah atau pendapatan pelanggan.

6. Eksperimen 2 — Overfitting (Polynomial Regression Derajat Tinggi)

Berikutnya kita melatih **Polynomial Regression** dengan derajat (degree) tinggi. Ini dicapai dengan membuat **Polynomial Features** (misal X, X^2, X^3, …) lalu menerapkan regresi linear pada fitur yang diperluas tersebut. Model jadi sangat fleksibel dan mampu mengikuti titik-titik data latih secara detail. Akibatnya sering terjadi **overfitting**: error training sangat

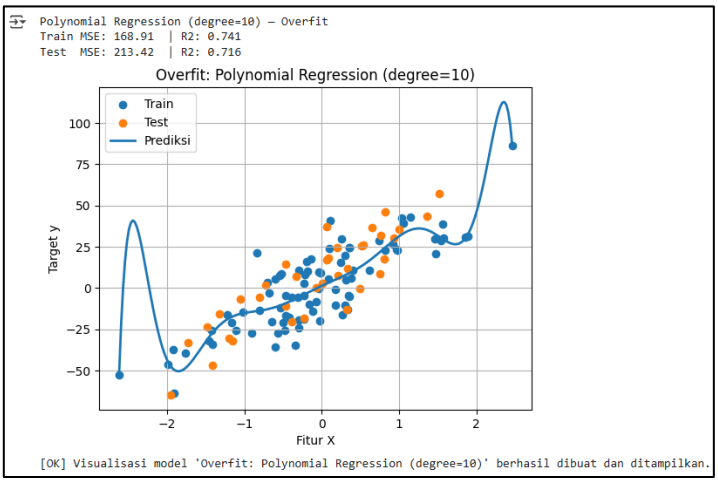
kecil, tetapi error testing memburuk karena model “menghafal” noise. Istilah kunci: **degree** adalah derajat polinomial; semakin tinggi degree, semakin kompleks bentuk kurva. Tujuannya memperlihatkan bagaimana model yang terlalu kompleks gagal melakukan generalisasi.

```
degree_overfit = 10  # Anda bisa bereksperimen: 8, 10, 12, 15
poly_overfit = Pipeline([
    ("poly", PolynomialFeatures(degree=degree_overfit, include_bias=False)),
    ("linreg", LinearRegression())
])
poly_overfit.fit(X_train, y_train)

y_pred_train = poly_overfit.predict(X_train)
y_pred_test  = poly_overfit.predict(X_test)

mse_train = mean_squared_error(y_train, y_pred_train)
mse_test  = mean_squared_error(y_test, y_pred_test)

r2_train = r2_score(y_train, y_pred_train)
r2_test  = r2_score(y_test, y_pred_test)

print(f"Polynomial Regression (degree={degree_overfit}) – Overfit")
print("Train MSE:", round(mse_train, 2), " | R2:", round(r2_train, 3))
print("Test  MSE:", round(mse_test, 2),  " | R2:", round(r2_test, 3))

plot_model_line(poly_overfit, X_train, y_train, X_test, y_test,
                title=f"Overfit: Polynomial Regression (degree={degree_overfit})")
```

Keterangan:

- degree = kompleksitas model (semakin tinggi → semakin detail).
- Pipeline = alur gabungan (transformasi → model).
- Overfitting biasanya terlihat dari Train error sangat kecil, Test error besar.
- Visualisasi membantu melihat bagaimana garis prediksi "berliku-liku" mengikuti data train.

Output:



Interpretasi:

1. Hasil Evaluasi Angka

- **Train MSE = 168.91 dan R^2 Train = 0.741** → Model tampak sangat baik pada data latih, error relatif kecil, R^2 cukup tinggi.

- **Test MSE = 213.42 dan R^2 Test = 0.716** → Walau R^2 test masih cukup tinggi, error test lebih besar daripada error train. Gap ini menandakan model mulai kehilangan kemampuan generalisasi.

2. Visualisasi Grafik

- Garis prediksi (biru) berliku-liku dengan tajam, mengikuti hampir setiap variasi kecil pada titik-titik data latih.
- Pola "bergelombang" ini adalah ciri khas polynomial regression dengan degree tinggi.
- Akibatnya, meskipun model dapat memprediksi data latih dengan sangat baik, ia menghafal noise (gangguan acak) dan variasi kecil. Pada data uji, pola yang "terlalu rumit" ini justru tidak sesuai, sehingga prediksi bisa meleset.

3. Kesimpulan Eksperimen

- Model dengan degree=10 terlalu kompleks untuk dataset sederhana ini.
- Perbedaan antara performa di train vs test (walau tidak ekstrem sekali) tetap menunjukkan indikasi overfitting.
- Garis prediksi tidak lagi sekadar mengikuti pola utama (hubungan linear), melainkan juga "mengejar" titik-titik individual.
- Dalam aplikasi nyata, model overfit sering terlihat hebat di data lama, tapi gagal ketika dipakai pada data baru.

4. Mengapa hasil kurva berubah saat mengubah degree_overfit?
Degree menentukan seberapa kompleks fungsi polinomial yang digunakan:

- Degree rendah (misal 2–3) → kurva sederhana, masih relatif halus, risiko underfit jika pola data lebih rumit.
- Degree menengah (misal 5–8) → kurva lebih fleksibel, bisa mengikuti kelengkungan pola data dengan cukup baik.
- Degree tinggi (10–15) → kurva jadi sangat berliku, mengikuti hampir semua titik data (termasuk noise) → overfitting.

Secara matematis, semakin tinggi degree, semakin banyak parameter (koefisien polinomial) yang harus dipelajari model. Semakin banyak parameter, semakin "liar" bentuk kurva yang bisa dihasilkan.

7. Eksperimen 3 — Mengurangi Overfitting dengan Regularization (Ridge)

Untuk mengurangi overfitting, digunakan Ridge Regression (L2 regularization) yang menambahkan penalti terhadap bobot berukuran besar agar model tidak terlalu mengikuti noise. Parameter alpha mengontrol seberapa kuat penalti diterapkan—semakin besar nilainya, semakin halus kurva prediksi. Dengan pengaturan alpha yang tepat, dicapai keseimbangan antara bias dan variance, sehingga performa model pada data latih dan uji menjadi lebih konsisten tanpa perlu mengganti algoritma dasar.

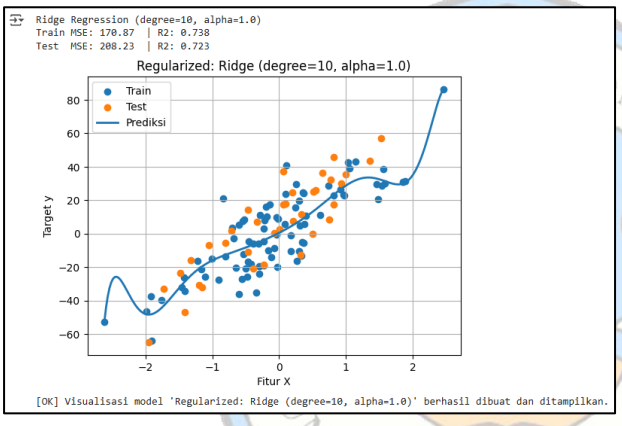
```
alpha = 1.0  # Coba variasi: 0.1, 1.0, 10.0
ridge_model = Pipeline([
    ("poly", PolynomialFeatures(degree=degree_overfit, include_bias=False)),
    ("ridge", Ridge(alpha=alpha, random_state=42))
])
ridge_model.fit(X_train, y_train)

y_pred_train = ridge_model.predict(X_train)
y_pred_test  = ridge_model.predict(X_test)

mse_train = mean_squared_error(y_train, y_pred_train)
mse_test  = mean_squared_error(y_test, y_pred_test)
```

```
r2_train = r2_score(y_train, y_pred_train)
r2_test  = r2_score(y_test, y_pred_test)

print(f"Ridge Regression (degree={degree_overfit}, alpha={alpha})")
print("Train MSE:", round(mse_train, 2), " | R2:", round(r2_train, 3))
print("Test  MSE:", round(mse_test, 2),  " | R2:", round(r2_test, 3))

plot_model_line(ridge_model, X_train, y_train, X_test, y_test,
                title=f"Regularized: Ridge (degree={degree_overfit}, alpha={alpha})")
```

Keterangan:

- Ridge Regression adalah cara mengurangi overfitting dengan memberi penalti pada koefisien besar.
- alpha adalah kunci pengendali seberapa kuat regularisasi diterapkan.
- Evaluasi Train vs Test menunjukkan apakah regularisasi berhasil membuat model lebih seimbang.

Output:



1. Hasil Evaluasi Angka

- **Train MSE = 170.87 | R² = 0.738** → Model cukup baik dalam menjelaskan data latih, tetapi tidak "sempurna". Ada sedikit penalti terhadap kompleksitas sehingga kurva tidak terlalu ekstrem.
- **Test MSE = 208.23 | R² = 0.723** → Nilai error pada data uji hampir seimbang dengan data latih. Artinya, model memiliki kemampuan generalisasi lebih baik dibandingkan Polynomial Regression tanpa regularisasi.
- **Kesimpulan awal**: Ridge berhasil "merapikan" kurva yang terlalu berliku-liku pada polynomial regression derajat tinggi, sehingga hasil train dan test lebih seimbang.

2. Visualisasi Grafik

- Garis prediksi (biru) masih mengikuti pola data dengan cukup fleksibel, tetapi tidak lagi terlalu ekstrim berliku-liku seperti pada polynomial regression murni (degree=10).
- Kurva tampak lebih halus, terutama di ujung kiri dan kanan, karena Ridge membatasi koefisien polinomial yang terlalu besar.
- Titik-titik data uji (oranye) lebih sesuai dengan prediksi garis, dibanding eksperimen overfitting sebelumnya.

3. Mengapa kurva berubah saat nilai alpha diubah?

- Alpha adalah parameter yang mengatur kekuatan regularisasi (penalti).
- Alpha kecil (misal 0.1) → penalti lemah. Model tetap kompleks, kurva bisa lebih berliku → cenderung mendekati polynomial regression biasa.
- Alpha sedang (misal 1.0) → penalti cukup kuat. Model tetap fleksibel, tetapi lebih terkendali. Bias dan variance relatif seimbang.
- Alpha besar (misal 10.0) → penalti sangat kuat. Koefisien polinomial besar ditekan mendekati nol. Hasilnya kurva semakin sederhana/halus, bahkan bisa kehilangan kelenturan sehingga mendekati garis linear lagi → risiko underfitting.

Dengan kata lain: Alpha = pengatur keseimbangan → kecil → model fleksibel tapi rawan overfit, besar → model lebih sederhana tapi rawan underfit.

Kesimpulan Eksperimen

- Ridge Regression terbukti mampu mengurangi overfitting pada polynomial regression derajat tinggi.
- Perbedaan error train dan test menjadi lebih kecil → indikasi model lebih stabil.
- Pemilihan nilai alpha sangat penting: terlalu kecil → tetap overfit, terlalu besar → bisa underfit.

8. Bandingkan dengan Lasso Regression (L1 regularization)

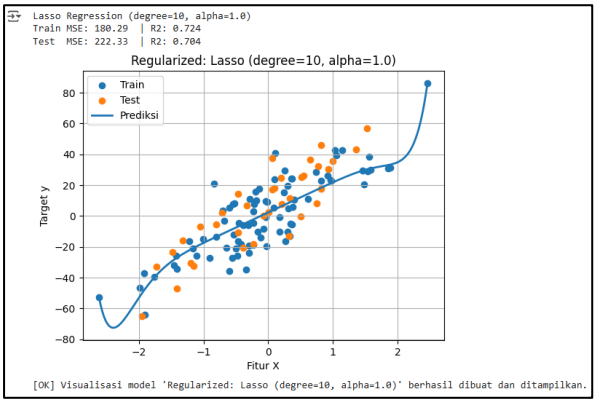
Sebagai alternatif, Lasso Regression (L1) digunakan untuk membandingkan efek regularisasi. Tidak seperti Ridge, Lasso dapat mengecilkan beberapa koefisien hingga nol, sehingga berfungsi juga sebagai seleksi fitur otomatis, khususnya pada model polinomial. Uji dengan berbagai nilai alpha menunjukkan bagaimana L1 dan L2 berbeda dalam mengatur kompleksitas dan kesederhanaan model, sekaligus memberi gambaran bahwa ada lebih dari satu pendekatan dalam regularisasi.

Perbedaan Ridge vs Lasso:

- Ridge (L2) → "mengecilkan semua koefisien".
- Lasso (L1) → "mematikan" beberapa koefisien menjadi 0 → seolah memilih fitur yang dianggap paling penting.

```
alpha_lasso = 1.0  # Coba variasi: 0.01, 0.05, 0.1
lasso_model = Pipeline([
    ("poly", PolynomialFeatures(degree=degree_overfit, include_bias=False)),
    ("lasso", Lasso(alpha=alpha_lasso, max_iter=10000, random_state=42))
])
lasso_model.fit(X_train, y_train)

y_pred_train = lasso_model.predict(X_train)
y_pred_test  = lasso_model.predict(X_test)

mse_train = mean_squared_error(y_train, y_pred_train)
mse_test  = mean_squared_error(y_test, y_pred_test)

r2_train = r2_score(y_train, y_pred_train)
r2_test  = r2_score(y_test, y_pred_test)

print(f"Lasso Regression (degree={degree_overfit}, alpha={alpha_lasso})")
print("Train MSE:", round(mse_train, 2), " | R2:", round(r2_train, 3))
print("Test  MSE:", round(mse_test, 2),  " | R2:", round(r2_test, 3))

plot_model_line(lasso_model, X_train, y_train, X_test, y_test,
                title=f"Regularized: Lasso (degree={degree_overfit}, alpha={alpha_lasso})")
```

Output:



1. Hasil Evaluasi Angka

- Train MSE = 180.29 | R^2 = 0.724 → Model cukup baik dalam menjelaskan data latih, meskipun error lebih besar dibanding Ridge. Hal ini karena Lasso "mengorbankan" beberapa koefisien menjadi nol sehingga model jadi lebih sederhana.
- Test MSE = 222.33 | R^2 = 0.704 → Error pada data uji masih cukup seimbang dengan data latih. Model mampu generalisasi, meskipun performa tidak setinggi polynomial tanpa regularisasi.

- **Kesimpulan awal**: Lasso berhasil mengurangi overfitting, tapi dengan konsekuensi kurva prediksi menjadi lebih sederhana karena beberapa koefisien fitur dipaksa ke nol.

2. Visualisasi Grafik

- Garis prediksi (biru) tampak lebih halus dibanding polynomial tanpa regularisasi, tetapi sedikit berbeda dengan Ridge.
- Pada beberapa bagian, kurva terlihat "ditarik" ke arah garis lurus, karena fitur polinomial tingkat tinggi dengan koefisien kecil diabaikan (ditekan jadi nol).
- Titik uji (oranye) sebagian besar masih mengikuti garis prediksi, meskipun ada penyebaran.

3. Mengapa kurva berubah saat nilai alpha_lasso diubah?

Alpha pada Lasso mengatur seberapa kuat penalti L1 diberikan pada koefisien:

- Alpha kecil (0.01, 0.05, 0.1) → Penalti lemah. Model masih mempertahankan banyak fitur polinomial. → Kurva lebih berliku-liku (mirip polynomial regression biasa). → Risiko overfitting lebih besar.
- Alpha sedang (1.0 seperti grafik ini) → Penalti cukup kuat. Beberapa koefisien polinomial kecil ditekan menjadi nol. → Kurva jadi lebih sederhana/halus. → Generalisasi lebih stabil di test set.
- Alpha besar (misalnya 10.0) → Penalti sangat kuat. Sebagian besar koefisien ditekan ke nol. → Model hampir kembali ke bentuk linear sederhana. → Risiko underfitting meningkat karena model terlalu disederhanakan.

4. Perbedaan Lasso vs Ridge dalam konteks ini

- Ridge (L2) → Mengecilkan semua koefisien, tetapi jarang menekan koefisien sampai nol. Hasilnya kurva tetap halus tapi mempertahankan semua fitur.
- Lasso (L1) → Menekan beberapa koefisien ke nol. Hasilnya kurva lebih sederhana, kadang kehilangan beberapa lengkungan. Bisa dianggap sebagai bentuk feature selection otomatis.

5. Kesimpulan Eksperimen

- Lasso Regression membantu mengurangi overfitting dengan cara memilih fitur polinomial yang dianggap penting dan membuang sisanya.
- Perubahan nilai alpha_lasso sangat berpengaruh terhadap bentuk kurva:
- Kecil → lebih rumit (rawan overfit)
- Sedang → seimbang
- Besar → terlalu sederhana (underfit)

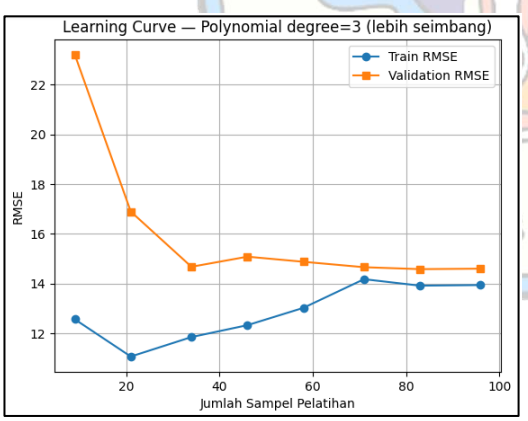
9. Learning Curve (Train vs Validation)

Terakhir, kita menggambar **learning curve**, yakni grafik metrik performa (di sini **RMSE**—akar dari MSE, makin kecil makin baik) pada **train** dan **validation** terhadap bertambahnya ukuran data latih. Kurva ini membantu mendeteksi pola **overfitting** (train error rendah, validation error tinggi dan menjauh) dan **underfitting** (keduanya tinggi dan tidak membaik). Kita memakai **cross-validation (CV)** di balik layar fungsi untuk memperoleh estimasi yang lebih stabil. Tujuan langkah ini adalah memberi alat diagnostik visual agar Anda tahu apakah perlu menambah data, menyederhanakan/meningkatkan model, atau menyesuaikan hyperparameter.

```python
def plot_learning_curve(estimator, X, y, title="Learning Curve", cv=5, scoring="neg_mean_squared_error"):
    train_sizes, train_scores, valid_scores = learning_curve(
        estimator, X, y, cv=cv, scoring=scoring, train_sizes=np.linspace(0.1, 1.0, 8), random_state=42
    )
    train_rmse = np.sqrt(-train_scores.mean(axis=1))
    valid_rmse = np.sqrt(-valid_scores.mean(axis=1))

    plt.figure()
    plt.plot(train_sizes, train_rmse, marker="o", label="Train RMSE")
    plt.plot(train_sizes, valid_rmse, marker="s", label="Validation RMSE")
    plt.title(title)
    plt.xlabel("Jumlah Sampel Pelatihan")
    plt.ylabel("RMSE")
    plt.legend()
    plt.grid(True)
    plt.show()

# Contoh: bandingkan LC untuk polynomial degree rendah vs tinggi
poly3 = Pipeline([("poly", PolynomialFeatures(degree=3, include_bias=False)), ("lin", LinearRegression())])
poly10 = Pipeline([("poly", PolynomialFeatures(degree=10, include_bias=False)), ("lin", LinearRegression())])

plot_learning_curve(poly3, X, y, title="Learning Curve – Polynomial degree=3 (lebih seimbang)")
plot_learning_curve(poly10, X, y, title="Learning Curve – Polynomial degree=10 (cenderung overfit)")
```

Keterangan:

- Learning curve digunakan untuk mendeteksi overfitting/underfitting.
- RMSE train vs validation membantu melihat apakah model terlalu rumit (overfit) atau terlalu sederhana (underfit).
- Perbedaan degree=3 (lebih seimbang) vs degree=10 (lebih kompleks) terlihat jelas di kurva.

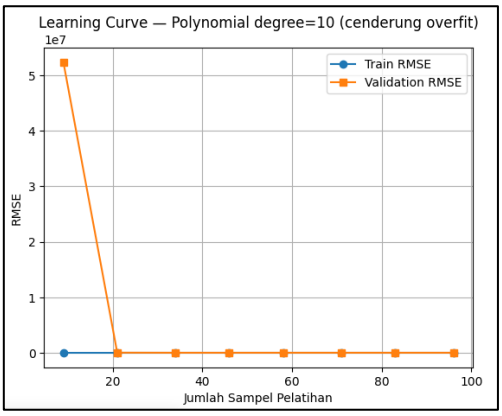


1. Learning Curve — Polynomial degree=3 (lebih seimbang)

- Train RMSE relatif rendah (sekitar 12–14) dan stabil meskipun jumlah data latih bertambah.
- Validation RMSE awalnya cukup tinggi (>22), tetapi turun drastis ketika jumlah data latih meningkat, lalu stabil di kisaran 14–15.
- Gap (selisih) antara train dan validation RMSE relatif kecil ketika jumlah data semakin besar.

Interpretasi:

- Model dengan degree=3 cukup fleksibel untuk menangkap pola utama tanpa terlalu kompleks.
- Tidak terjadi gejala overfitting yang signifikan karena performa train dan validation konsisten.
- Model ini bisa dikatakan cukup seimbang (balanced bias-variance tradeoff), sehingga cocok digunakan untuk generalisasi.



2. Learning Curve — Polynomial degree=10 (cenderung overfit)

- Train RMSE terlihat sangat kecil (hampir nol) sejak awal → tanda bahwa model hampir "menghafal" data latih.
- Validation RMSE awalnya sangat tinggi (bahkan >50 juta) lalu turun tajam, tetapi tetap jauh berbeda dari train RMSE.
- Gap antara train vs validation RMSE sangat besar, bahkan setelah data bertambah.

Interpretasi:

- Model dengan degree=10 terlalu kompleks untuk dataset ini, sehingga overfitting:
- Sangat bagus di train (error hampir nol).
- Sangat buruk di validation (error besar, tidak stabil).
- Kurva ini menunjukkan model tidak bisa melakukan generalisasi dengan baik meskipun data latih bertambah.

3. Kesimpulan Umum

- Degree=3 → model sederhana namun cukup akurat, hasil train dan validation relatif konsisten → model seimbang.
- Degree=10 → model terlalu rumit, hasil train nyaris sempurna tapi validation buruk → model overfitting.
- Jika train & validation error tinggi dan sejajar → underfitting.
- Jika train error rendah, validation error tinggi → overfitting.
- Jika train & validation error rendah dan sejajar → model seimbang (good generalization).

10. Tugas Praktikum

a) Ubah nilai **degree_overfit** ke 2, 5, 15 dan catat perbedaan **MSE** dan **R²** untuk **train** serta **test**. Jelaskan fenomena yang Anda lihat.

b) Ganti **Ridge** dengan **Lasso** pada eksperimen regularisasi. Coba beberapa nilai **alpha** (misalnya 0.01, 0.05, 0.1). Bandingkan hasilnya dan simpulkan.

c) Jalankan fungsi *learning curve* untuk model yang Anda anggap terbaik. Apakah kurva train dan validasi tampak **seimbang**? Jelaskan indikasi overfitting/underfitting berdasarkan grafik.

d) Diskusikan dalam kelompok: pada konteks bisnis digital, dalam kasus apa **underfitting** lebih berbahaya daripada **overfitting**, dan sebaliknya? Sertakan contoh nyata.

D. LATIHAN DAN TUGAS

1. Latihan

a) Buatlah grafik **learning curve** dari dua model berbeda: satu yang underfit (misalnya regresi linear sederhana) dan satu yang overfit (misalnya polynomial regression derajat tinggi).

b) Identifikasi pada grafik tersebut bagaimana tren error/akurasi pada data latih dan data uji. Tuliskan kesimpulan singkat mengenai titik di mana model mulai mengalami overfitting.

2. Tugas

a) Gunakan dataset dummy sederhana (misalnya prediksi penjualan) dan uji dengan tiga jenis model:

- Linear Regression (underfit)
- Polynomial Regression orde tinggi (overfit)
- Regularized Regression (Ridge atau Lasso)

b) Bandingkan performa ketiga model tersebut dengan melihat nilai error pada data latih dan data uji.

c) Susun laporan ringkas berisi tabel perbandingan hasil, grafik visualisasi, serta kesimpulan kelompok mengenai model mana yang paling seimbang.

3. Diskusi

a) Menurut Anda, dalam konteks **bisnis digital**, mana yang lebih berbahaya: underfitting atau overfitting?

b) Berikan contoh nyata:

- **Underfitting** → misalnya model prediksi rekomendasi produk yang terlalu sederhana sehingga tidak mampu memberikan saran relevan.
- **Overfitting** → misalnya model deteksi fraud yang terlalu menyesuaikan data lama sehingga gagal mengenali pola penipuan baru.

MODUL IX
ANALISIS DATA BISNIS DENGAN ML

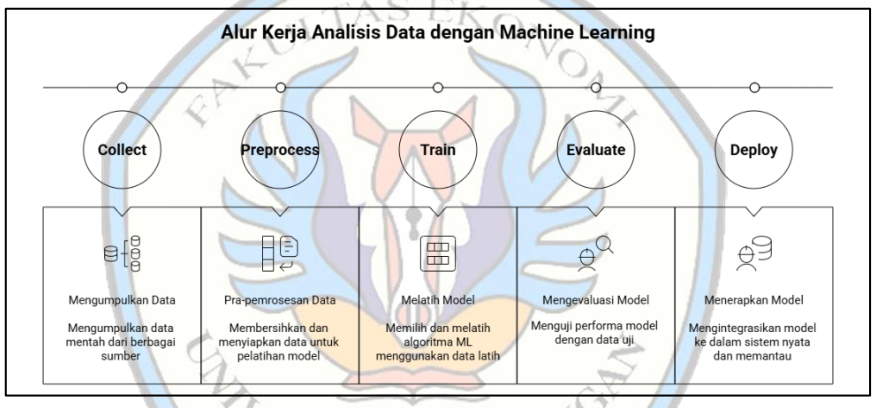
A. CAPAIAN PEMBELAJARAN

1) Mahasiswa memahami workflow analisis data dengan ML.
2) Mahasiswa mampu melakukan studi kasus prediksi retensi pelanggan.
3) Mahasiswa mengenal integrasi ML dengan tools bisnis digital.
4) Mahasiswa mampu mengimplementasikan mini project sederhana berbasis bisnis

B. DASAR TEORI

1. Workflow Analisis Data dengan ML

Dalam dunia bisnis digital, Machine Learning tidak hanya berhenti pada pembuatan model. Proses analisis data dengan ML memiliki alur kerja yang sistematis agar hasilnya dapat digunakan secara nyata untuk mendukung pengambilan keputusan. Secara umum, workflow ini dapat diringkas dalam lima tahap utama:



a) Collect (Mengumpulkan Data)

Tahap pertama adalah mengumpulkan data dari berbagai sumber, seperti transaksi e-commerce, interaksi media sosial, log aplikasi, hingga data survei pelanggan. Data yang dikumpulkan harus relevan dengan permasalahan bisnis yang ingin dipecahkan.

b) Preprocess (Pra-pemrosesan Data)

Data mentah biasanya masih berantakan, sehingga perlu dibersihkan dan dipersiapkan terlebih dahulu. Langkah ini meliputi penanganan missing values, duplikasi, normalisasi, encoding data kategorikal, hingga feature engineering. Kualitas preprocessing akan sangat menentukan akurasi model.

c) Train (Melatih Model)

Pada tahap ini, algoritma ML dipilih sesuai dengan tujuan bisnis. Misalnya, regresi untuk prediksi penjualan, klasifikasi untuk deteksi churn pelanggan, atau clustering untuk segmentasi pasar. Model kemudian dilatih menggunakan data latih agar dapat mengenali pola.

d) Evaluate (Mengevaluasi Model)

Setelah dilatih, model diuji dengan data uji untuk mengukur performanya. Metrik yang digunakan bergantung pada kasus, misalnya akurasi, precision-recall, F1-score, atau RMSE. Evaluasi penting agar model tidak hanya baik di data latih, tetapi juga generalis di data baru.

e) Deploy (Menerapkan Model)

Tahap akhir adalah implementasi model ke dalam sistem nyata, seperti dashboard analitik, aplikasi rekomendasi, atau chatbot layanan pelanggan. Di tahap ini, model harus dipantau terus-menerus karena data bisnis bisa berubah seiring waktu.

Selain memahami alur kerja ini, penting juga mengenal **peran tim** yang terlibat dalam pipeline ML:

- **Data Engineer**: bertugas menyiapkan infrastruktur data, mulai dari proses ekstraksi, transformasi, hingga penyimpanan (ETL).
- **Data Scientist**: fokus pada eksplorasi data, pembuatan model ML, serta interpretasi hasil dalam konteks bisnis.
- **ML Engineer**: mengintegrasikan model yang sudah jadi ke dalam sistem produksi, serta memastikan model dapat berjalan efisien dan skalabel.

Dengan pemahaman ini, mahasiswa dapat melihat bahwa ML dalam bisnis bukan hanya soal coding algoritma, tetapi juga soal mengelola alur kerja secara menyeluruh agar model benar-benar memberi dampak nyata pada keputusan bisnis.

2. Studi Kasus: Prediksi Retensi Pelanggan

Salah satu penerapan Machine Learning yang paling relevan dalam bisnis digital adalah **prediksi retensi pelanggan**. Retensi mengacu pada kemampuan sebuah perusahaan untuk mempertahankan pelanggan agar terus menggunakan produk atau layanannya. Sebaliknya, **churn** adalah kondisi ketika pelanggan berhenti menggunakan layanan, misalnya tidak lagi berlangganan aplikasi streaming atau tidak kembali berbelanja di sebuah toko online.

Definisi Retensi dan Churn

- **Retensi Pelanggan** → kondisi di mana pelanggan tetap aktif, melakukan pembelian ulang, atau memperpanjang langganan.
- **Churn Pelanggan** → kondisi pelanggan berhenti, misalnya tidak lagi membuka aplikasi dalam jangka waktu lama atau memutuskan berhenti berlangganan layanan premium.

Faktor-Faktor yang Mempengaruhi Retensi

Beberapa aspek yang sangat berpengaruh terhadap perilaku pelanggan untuk tetap bertahan di sebuah layanan antara lain:

- **Harga**: pelanggan sensitif terhadap kenaikan harga, terutama jika pesaing menawarkan produk serupa dengan biaya lebih rendah.
- **Promo & Insentif**: diskon, program loyalitas, atau cashback dapat meningkatkan kemungkinan pelanggan untuk tetap aktif.
- **Kepuasan Pelanggan**: pengalaman pengguna yang baik, layanan customer service yang responsif, dan kualitas produk/jasa yang konsisten menjadi faktor kunci retensi.
- **Kenyamanan & Fitur**: semakin mudah dan praktis sebuah aplikasi atau layanan digunakan, semakin besar kemungkinan pelanggan bertahan.

Mengapa Prediksi Retensi Penting bagi Bisnis Digital

Dalam persaingan bisnis digital yang sangat ketat, mempertahankan pelanggan jauh lebih murah daripada mencari pelanggan baru. Dengan prediksi retensi, perusahaan bisa:

a) **Mengidentifikasi pelanggan berisiko churn** lebih awal, sehingga dapat diberikan penawaran atau intervensi khusus.

b) **Mengoptimalkan strategi pemasaran** dengan fokus pada pelanggan yang paling berharga (high-value customers).
c) **Meningkatkan profitabilitas jangka panjang**, karena pelanggan yang loyal biasanya lebih sering bertransaksi dan memberikan rekomendasi positif.

Dengan demikian, analisis retensi pelanggan melalui Machine Learning tidak hanya membantu perusahaan memahami perilaku konsumennya, tetapi juga menjadi dasar pengambilan keputusan strategis untuk mempertahankan keunggulan kompetitif di era digital.

Contoh Mini Dataset Retensi Pelanggan

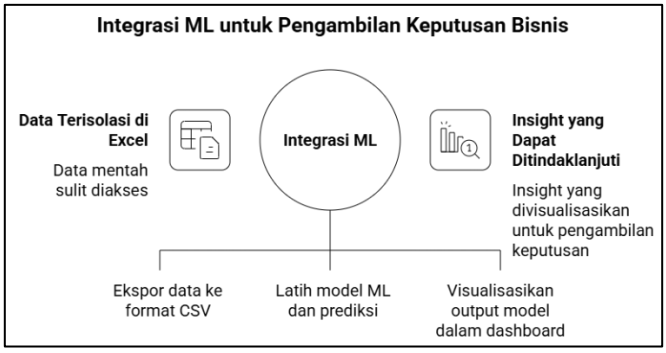
ID Pelanggan	Jumlah Transaksi	Rata2 NilaiBeli	Diskon Rata2	Hari Semjak Transaksi Terakhir	Status Loyal
1001	15	300000	0.10	5	1 (Loyal)
1002	3	120000	0.05	120	0 (Churn)
…	…	…	…	…	…
1009	10	220000	0.12	20	1 (Loyal)
1010	4	150000	0.08	100	0 (Churn)

Keterangan:

- Jumlah_Transaksi → total transaksi yang pernah dilakukan.
- Rata2_Nilai_Beli → rata-rata nilai belanja per transaksi (Rp).
- Diskon_Rata2 → rata-rata diskon yang digunakan pelanggan (0–1).
- Hari_Semjak_Transaksi_Terakhir → ukuran *recency*, makin besar artinya makin lama tidak bertransaksi.
- Status_Loyal → label target:
  - 1 (Loyal) → pelanggan masih aktif / berpotensi bertahan.
  - 0 (Churn) → pelanggan tidak aktif / kemungkinan besar berhenti.

3. Integrasi dengan Tools Bisnis Digital

Dalam praktik nyata, hasil analisis Machine Learning tidak berhenti di kode Python semata. Agar dapat memberikan nilai tambah bagi bisnis, hasil model harus diintegrasikan ke dalam alur kerja yang lebih praktis dan mudah digunakan oleh pengambil keputusan. Integrasi ini umumnya mengikuti alur sederhana: Excel → Python (ML model) → Dashboard (visualisasi).



a) Excel sebagai titik awal data

Banyak perusahaan, khususnya UMKM dan bisnis digital kecil, masih menggunakan Excel atau Google Sheets untuk mencatat transaksi, data pelanggan, maupun laporan penjualan. Data dari Excel dapat diekspor ke format CSV, lalu diolah lebih lanjut dengan Python untuk membangun model Machine Learning.

b) Python sebagai mesin analitik

Setelah data siap, Python digunakan untuk menjalankan proses *preprocessing*, melatih model ML, serta menghasilkan prediksi (misalnya siapa pelanggan yang berisiko churn). Output Python dapat berupa tabel hasil prediksi atau file baru yang siap dipakai untuk pelaporan.

c) Dashboard sebagai media komunikasi hasil

Agar hasil analisis dapat dipahami dengan cepat oleh manajer atau tim non-teknis, output model perlu divisualisasikan dalam bentuk dashboard interaktif. Beberapa tools populer yang dapat digunakan antara lain:

- **Google Data Studio** → terhubung langsung dengan Google Sheets atau file CSV untuk membuat laporan interaktif.
- **Power BI** → cocok untuk perusahaan menengah hingga besar, dengan integrasi berbagai sumber data.
- **Streamlit** → library Python yang memungkinkan pembuatan aplikasi dashboard interaktif langsung dari notebook atau script Python.

Dengan alur ini, proses analisis data bisnis menjadi lebih menyeluruh: data mentah yang semula hanya berupa tabel Excel dapat ditransformasikan menjadi insight yang divisualisasikan dalam dashboard, sehingga memudahkan pengambilan keputusan.

C. PRAKTIKUM 1 (Retensi Pelanggan - Workflow ML Dasar dengan Dataset Kecil)

Tujuan Praktikum

a) Menerapkan workflow analisis data bisnis dengan ML.
b) Melatih model sederhana untuk prediksi retensi pelanggan (churn vs loyal).
c) Membuat visualisasi hasil prediksi agar mudah dipahami dalam konteks bisnis.

Pada mini project ini, kita melakukan analisis retensi pelanggan dengan workflow machine learning sederhana. Dimulai dari import dataset pelanggan (mini_dataset_retensi_pelanggan_100.csv), kemudian dilakukan tahap preprocessing untuk menyiapkan data agar sesuai dengan kebutuhan model. Selanjutnya, kita melatih model Logistic Regression sebagai algoritma klasifikasi biner untuk memprediksi status **loyal** atau **churn**. Hasil prediksi dievaluasi melalui **confusion matrix** agar mudah dilihat perbandingan prediksi vs data aktual. Sebagai tambahan, kita juga dapat menampilkan distribusi probabilitas prediksi untuk memahami tingkat keyakinan model terhadap keputusan yang diambil.

1. Setup & Library

```
import numpy as np              # operasi numerik
import pandas as pd             # olah data tabular
import matplotlib.pyplot as plt # visualisasi grafik
# Import fungsi train_test_split untuk membagi data menjadi data latih & data uji
from sklearn.model_selection import train_test_split
# Import StandardScaler untuk melakukan standarisasi fitur (mean=0, std=1)
from sklearn.preprocessing import StandardScaler
# Import LogisticRegression sebagai algoritma klasifikasi biner (loyal vs churn)
from sklearn.linear_model import LogisticRegression
# Import metrik evaluasi untuk menilai performa model
from sklearn.metrics import accuracy_score, classification_report, confusion_matrix

np.random.seed(42)  # agar hasil acak konsisten
print("Library siap dipakai.")
```

```
Library siap dipakai.
```

2. Import Dataset Pelanggan

Membaca dataset pelanggan yang berisi informasi transaksi, nilai belanja, diskon, dan status loyal/churn untuk dianalisis.

```python
import pandas as pd
import numpy as np

# Set random seed agar hasil konsisten
np.random.seed(42)

# Jumlah data yang ingin dibuat
N = 100

# Buat dataset dummy
data = {
    "ID_Pelanggan": np.arange(1001, 1001 + N),
    "Jumlah_Transaksi": np.random.randint(1, 30, size=N),  # antara 1 - 30 transaksi
    "Rata2_Nilai_Beli": np.random.randint(50_000, 500_000, size=N),  # rata-rata belanja Rp 50k - 500k
    "Diskon_Rata2": np.round(np.random.uniform(0, 0.3, size=N), 2),  # diskon antara 0 - 30%
    "Hari_Semjak_Transaksi_Terakhir": np.random.randint(1, 180, size=N),  # recency 1-180 hari
    "Status_Loyal": np.random.choice([0, 1], size=N, p=[0.4, 0.6])  # 40% churn, 60% loyal
}

df = pd.DataFrame(data)

# Simpan ke CSV
output_file = "mini_dataset_retensi_pelanggan_100.csv"
df.to_csv(output_file, index=False)

print("Dataset berhasil dibuat:", output_file)
print(df.head())
```

Outputnya seperti berikut:

```
Dataset berhasil dibuat: mini_dataset_retensi_pelanggan_100.csv
   ID_Pelanggan  Jumlah_Transaksi  Rata2_Nilai_Beli  Diskon_Rata2  \
0          1001                 7             90397          0.03
1          1002                20            307750          0.14
2          1003                29            403531          0.07
3          1004                15            313160          0.12
4          1005                11            105591          0.26

   Hari_Semjak_Transaksi_Terakhir  Status_Loyal
0                              68             0
1                              33             0
2                             142             1
3                              21             1
4                              48             1
```

3. Preprocessing Data Pelanggan

- Pisahkan fitur dan target.
- Lakukan scaling untuk fitur numerik.

```python
# Pisahkan fitur dan target
X = df.drop(columns=["ID_Pelanggan", "Status_Loyal"])
y = df["Status_Loyal"]

# Train-Test Split (hindari data leakage)
X_train, X_test, y_train, y_test = train_test_split(
    X, y, test_size=0.3, random_state=42, stratify=y
)

# Standarisasi fitur numerik
scaler = StandardScaler()
X_train_scaled = scaler.fit_transform(X_train)
X_test_scaled = scaler.transform(X_test)

print("Ukuran data latih:", X_train.shape, "| data uji:", X_test.shape)
```

```
Ukuran data latih: (70, 4) | data uji: (30, 4)
```

4. Melatih Model ML (Logistic Regression)

Kita gunakan Logistic Regression karena interpretasinya mudah dan cocok untuk kasus klasifikasi biner (loyal vs churn).

```
model = LogisticRegression()
model.fit(X_train_scaled, y_train)

# Prediksi
y_pred = model.predict(X_test_scaled)

# Evaluasi
print("Akurasi:", accuracy_score(y_test, y_pred))
print(classification_report(y_test, y_pred, digits=3))
```

```
Akurasi: 0.5666666666666667
              precision    recall  f1-score   support

           0      0.000     0.000     0.000        13
           1      0.567     1.000     0.723        17

    accuracy                          0.567        30
   macro avg      0.283     0.500     0.362        30
weighted avg      0.321     0.567     0.410        30
```

Interpretasi:

Akurasi rendah (56%)

Dari 30 data uji, model hanya benar di sekitar setengah kasus. Ini menunjukkan model Logistic Regression belum mampu menangkap pola yang baik dari data latih.

Kelas 0 (Churn) tidak pernah diprediksi

- Lihat baris 0: precision, recall, f1-score semuanya 0.000.
- Artinya, model selalu memprediksi kelas 1 (Loyal), tidak pernah memprediksi kelas 0.
- Ini menyebabkan warning: “Precision is ill-defined and being set to 0.0 in labels with no predicted samples”.

Kelas 1 (Loyal) sangat dominan

- Recall kelas 1 = 1.000 (semua data loyal terprediksi benar).
- Tapi precision hanya 0.567, artinya ada cukup banyak pelanggan yang sebenarnya churn tapi ikut diprediksi loyal.

Kemungkinan penyebab

- Dataset kecil (100 baris) → variasi pola churn terbatas.
- Imbalance data → mungkin proporsi loyal lebih besar dari churn (misalnya 60% vs 40%).
- Logistic Regression default → kadang condong memprediksi kelas mayoritas.

Interpretasi Hasil

- Model gagal mendeteksi pelanggan churn (kelas 0) → semua dianggap loyal.
- Akurasi 56% setara dengan “asal tebak selalu loyal” (baseline).
- Recall kelas loyal tinggi (100%), artinya semua pelanggan loyal berhasil dikenali, tapi tidak ada pelanggan churn yang terdeteksi.
- Risiko bisnis: model seperti ini berbahaya, karena perusahaan tidak bisa mengenali pelanggan yang berisiko berhenti (churn).

Implikasi & Next Step

- Gunakan dataset lebih besar (1000 baris dari generator, bukan 100).
- Tangani imbalance: coba teknik seperti:
- class_weight="balanced" pada Logistic Regression.

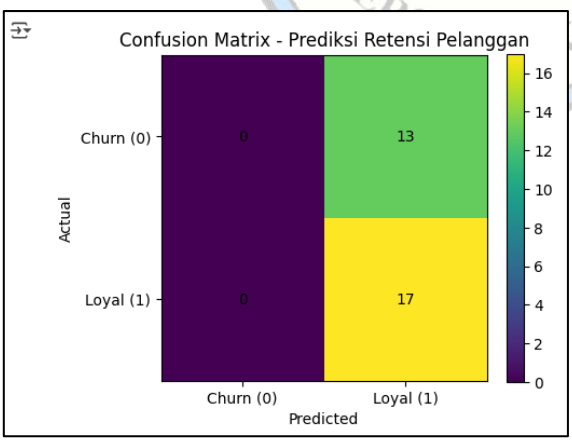
- Oversampling (SMOTE) atau undersampling.
- Uji model lain: Decision Tree, Random Forest, atau KNN yang mungkin lebih adaptif terhadap distribusi data.

5. Visualisasi Hasil Prediksi

Gunakan confusion matrix dan grafik sederhana untuk menunjukkan performa model. Menampilkan perbandingan hasil prediksi model dengan data aktual, sehingga terlihat jumlah prediksi benar maupun salah pada tiap kelas.

```
cm = confusion_matrix(y_test, y_pred)
classes = ["Churn (0)", "Loyal (1)"]

plt.figure(figsize=(5,4))
plt.imshow(cm, interpolation='nearest')
plt.title("Confusion Matrix - Prediksi Retensi Pelanggan")
plt.colorbar()
tick_marks = np.arange(len(classes))
plt.xticks(tick_marks, classes, rotation=0)
plt.yticks(tick_marks, classes)

# Tampilkan angka di setiap kotak
thresh = cm.max() / 2.
for i in range(cm.shape[0]):
    for j in range(cm.shape[1]):
        plt.text(j, i, format(cm[i, j], 'd'),
                 ha="center", va="center")

plt.ylabel("Actual")
plt.xlabel("Predicted")
plt.tight_layout()
plt.show()
```

Outputnya seperti berikut:



Interpretasi Confusion Matrix

Dari gambar, isi confusion matrix adalah:

- Churn (0) → Prediksi Churn (0): 0
- Churn (0) → Prediksi Loyal (1): 13
- Loyal (1) → Prediksi Churn (0): 0
- Loyal (1) → Prediksi Loyal (1): 17

Apa artinya?

- Model tidak pernah memprediksi churn (kelas 0). Semua pelanggan diprediksi sebagai loyal (kelas 1), meskipun ada 13 pelanggan yang sebenarnya churn.

- Recall kelas Loyal sangat tinggi (100%). Semua pelanggan yang loyal berhasil dikenali dengan benar.
- Recall kelas Churn = 0%. Tidak ada satupun pelanggan churn yang terdeteksi → model gagal mendeteksi pelanggan berisiko.
- Akurasi keseluruhan 56% Dari total 30 data, hanya 17 yang benar → akurasi rendah dan tidak lebih baik dibanding baseline sederhana (selalu prediksi mayoritas).

Kesimpulan

- Model bias ke kelas mayoritas (Loyal) sehingga mengabaikan kelas minoritas (Churn).
- Dalam konteks bisnis, hasil ini tidak bermanfaat karena tujuan utama analisis retensi adalah menemukan pelanggan yang berisiko churn.
- Perlu strategi perbaikan seperti: Menambah jumlah data agar distribusi lebih seimbang. Menggunakan class_weight="balanced" pada Logistic Regression. Mencoba algoritma lain (Decision Tree, Random Forest, KNN) yang lebih adaptif terhadap ketidakseimbangan kelas. Oversampling/SMOTE pada kelas minoritas.

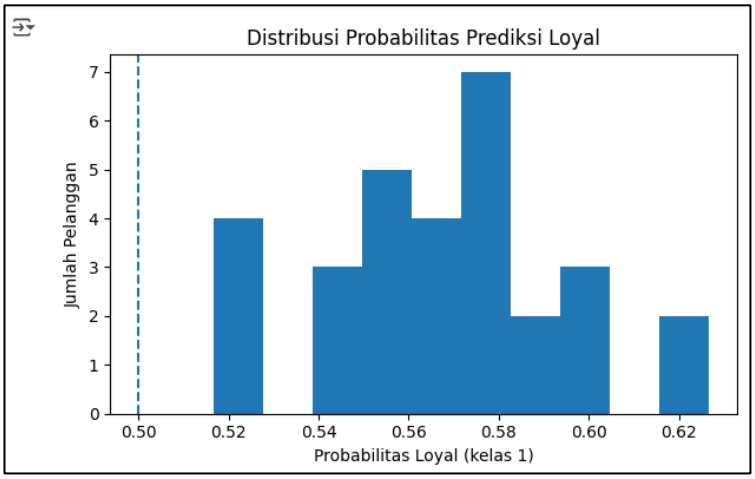
Jadi, singkatnya: Model hanya mampu mengenali pelanggan loyal, tetapi gagal total dalam mendeteksi pelanggan churn. Ini berbahaya untuk bisnis karena justru pelanggan churn yang paling perlu ditangani.

6. (Opsional) Dashboard Sederhana dengan Matplotlib

Memvisualisasikan tingkat probabilitas prediksi (keyakinan model) terhadap pelanggan yang diprediksi loyal maupun churn.

```
# Probabilitas prediksi kelas 1 (Loyal)
if hasattr(model, "predict_proba"):
    y_proba = model.predict_proba(X_test_scaled)[:, 1]

    plt.figure(figsize=(6,4))
    plt.hist(y_proba, bins=10)
    plt.axvline(0.5, linestyle="--")
    plt.title("Distribusi Probabilitas Prediksi Loyal")
    plt.xlabel("Probabilitas Loyal (kelas 1)")
    plt.ylabel("Jumlah Pelanggan")
    plt.tight_layout()
    plt.show()
else:
    print("Model tidak menyediakan predict_proba.")
```

Outputnya Seperti berikut:



Interpretasi Distribusi Probabilitas Prediksi

- Grafik menunjukkan distribusi probabilitas prediksi loyal (kelas 1) untuk data uji.
- Hampir semua nilai probabilitas berada di rentang 0.52 – 0.62, relatif dekat dengan ambang batas default 0.5 (garis vertikal putus-putus).
- Tidak ada probabilitas yang tinggi sekali (misalnya >0.8), sehingga model kurang percaya diri dalam membuat prediksi.
- Karena semua nilai berada di atas 0.5, model akhirnya selalu mengklasifikasikan semua pelanggan sebagai loyal (kelas 1).

Apa artinya?

- **Model terlalu "netral"** → Prediksi probabilitas tidak menunjukkan perbedaan yang jelas antara pelanggan churn dan loyal.
- **Threshold 0.5 tidak membantu** → Karena semua probabilitas >0.5, maka semua diprediksi loyal.
- **Kurangnya separasi kelas** → Menandakan bahwa fitur yang digunakan (jumlah transaksi, rata-rata nilai beli, diskon, recency) tidak cukup informatif untuk memisahkan churn vs loyal dalam dataset kecil ini.

7. Diskusi

a) Bagaimana performa model Logistic Regression pada dataset mini ini?
b) Apakah scaling memengaruhi hasil prediksi?
c) Bagaimana jika threshold probabilitas diubah (misalnya 0.4 atau 0.6)?

D. PRAKTIKUM 2 (Retensi Pelanggan - Pipeline Numerik & Kategorikal di Dataset Besar)

Tujuan Praktikum

- Mahasiswa mampu memahami tahapan analisis retensi pelanggan dengan dataset yang lebih besar dan kompleks.
- Mahasiswa mampu melakukan preprocessing data campuran (numerik & kategorikal) menggunakan ColumnTransformer.
- Mahasiswa mampu melatih dan mengevaluasi model klasifikasi (Logistic Regression) untuk memprediksi status loyal vs churn.
- Mahasiswa dapat menginterpretasikan hasil evaluasi model melalui confusion matrix dan distribusi probabilitas prediksi.
- Mahasiswa terbiasa dengan workflow end-to-end ML yang lebih realistis: mulai dari membaca data, preprocessing, training, hingga evaluasi model.

Praktikum ini merupakan lanjutan dari analisis retensi pelanggan, dengan menggunakan dataset yang lebih besar (±1000 baris) dan memiliki kombinasi fitur numerik serta kategorikal (Kota). Pada versi ini, preprocessing dilakukan dengan pipeline campuran yang mencakup imputasi, standarisasi untuk data numerik, serta encoding untuk data kategorikal. Dengan alur ini, mahasiswa dapat merasakan workflow machine learning yang lebih mendekati kondisi nyata di industri, di mana data yang digunakan biasanya lebih kompleks dan beragam.

1. Mount Google Drive & Import Dataset

Tahap pertama adalah menghubungkan Google Drive ke Colab agar dataset dapat diakses langsung dari penyimpanan mahasiswa. Setelah itu, dataset pelanggan berisi ±1000 baris data retensi dimuat menggunakan Pandas untuk kemudian ditampilkan sekilas, sehingga mahasiswa dapat memastikan data berhasil diimpor dengan benar.

```python
from google.colab import drive
drive.mount('/content/drive')

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

# Info lingkungan
print(f"Python : {sys.version.split()[0]}")
print(f"Pandas : {pd.__version__}")
print(f"NumPy  : {np.__version__}")

# Path dataset (ubah jika perlu)
csv_path = "/content/drive/MyDrive/ColabNotebooks/Modul9/dataset_retensi_1000_BD01.csv"

df = pd.read_csv(csv_path)
print("Dataset dimuat dari:", csv_path)
df.head()
```

Outputnya seperti berikut:

```
Drive already mounted at /content/drive; to attempt to forcibly remount, call drive.mount("/content/drive", force_remount=True).
Python : 3.12.11
Pandas : 2.2.2
NumPy  : 2.0.2
Dataset dimuat dari: /content/drive/MyDrive/ColabNotebooks/Modul9/dataset_retensi_1000_BD01.csv
```

	ID_Pelanggan	Kota	Jumlah_Transaksi	Rata2_Nilai_Beli	Diskon_Rata2	Hari_Semjak_Transaksi_Terakhir	Status_Loyal
0	430492	Bandung	12.0	292269.0	0.155	55	1
1	430493	Bandung	9.0	121755.0	0.253	106	1
2	430494	Yogyakarta	6.0	234982.0	0.141	80	1
3	430495	Bandung	8.0	241532.0	0.055	21	1
4	430496	Yogyakarta	6.0	332063.0	0.063	61	1

2. Quick EDA (cek struktur & missing values)

Tahap ini digunakan untuk mengenali struktur awal dataset, termasuk ukuran data, tipe kolom, serta jumlah nilai hilang (missing values) di setiap fitur. Dengan eksplorasi sederhana ini, mahasiswa bisa memahami kondisi data mentah dan menentukan langkah preprocessing yang diperlukan sebelum masuk ke model ML.

```python
# Menampilkan jumlah baris dan kolom dataset
print("Ukuran data:", df.shape)

# Menampilkan nama kolom beserta tipe datanya (numerik/kategorikal)
print("\nKolom & tipe data:")
print(df.dtypes)

# Mengecek jumlah nilai hilang (missing values) di setiap kolom
print("\nJumlah missing per kolom:")
print(df.isna().sum())

# Menampilkan 5 baris pertama dataset sebagai contoh isi data
df.head()
```

Outputnya seperti berikut:

```
Ukuran data: (1010, 7)

Kolom & tipe data:
ID_Pelanggan                    int64
Kota                           object
Jumlah_Transaksi              float64
Rata2_Nilai_Beli              float64
Diskon_Rata2                  float64
Hari_Semjak_Transaksi_Terakhir  int64
Status_Loyal                    int64
dtype: object

Jumlah missing per kolom:
ID_Pelanggan                    0
Kota                            0
Jumlah_Transaksi               29
Rata2_Nilai_Beli               19
Diskon_Rata2                   20
Hari_Semjak_Transaksi_Terakhir  0
Status_Loyal                    0
dtype: int64
```

	ID_Pelanggan	Kota	Jumlah_Transaksi	Rata2_Nilai_Beli	Diskon_Rata2	Hari_Semjak_Transaksi_Terakhir	Status_Loyal
0	430492	Bandung	12.0	292269.0	0.155	55	1
1	430493	Bandung	9.0	121755.0	0.253	106	1
2	430494	Yogyakarta	6.0	234982.0	0.141	80	1
3	430495	Bandung	8.0	241532.0	0.055	21	1
4	430496	Yogyakarta	6.0	332063.0	0.063	61	1

Deskripsi Output Quick EDA:

Hasil output menunjukkan bahwa dataset memiliki **1010 baris dan 7 kolom**, dengan tipe data campuran: numerik (int64/float64) dan kategorikal (Kota). Dari pengecekan missing values, terdapat beberapa nilai kosong di kolom **Jumlah_Transaksi (29), Rata2_Nilai_Beli (19), dan Diskon_Rata2 (20)**, sedangkan kolom lainnya lengkap. Pada preview 5 baris pertama, terlihat informasi pelanggan seperti **ID, Kota, jumlah transaksi, rata-rata nilai belanja, diskon rata-rata, recency (hari sejak transaksi terakhir), dan status loyal (0=churn, 1=loyal)**.

3. Preprocessing & Split

Pada tahap ini dilakukan pemisahan fitur (X) dan target (y), kemudian membagi dataset menjadi data latih dan data uji. Karena dataset mengandung fitur numerik dan kategorikal, digunakan ColumnTransformer dengan pipeline khusus: numerik diimputasi median dan distandarisasi, sedangkan kategorikal diimputasi modus lalu diencode menggunakan OneHotEncoder. Hal ini memastikan semua data berada pada format numerik yang siap diproses model.

- Tangani **numerik** (imputasi median + StandardScaler).
- Tangani **kategorikal** (imputasi modus + OneHotEncoder).
- Gunakan **ColumnTransformer** agar treatment sesuai tipe kolom.

```python
# Import library untuk split data, preprocessing, pipeline, dan encoding
from sklearn.model_selection import train_test_split
from sklearn.compose import ColumnTransformer
from sklearn.pipeline import Pipeline
from sklearn.impute import SimpleImputer
from sklearn.preprocessing import OneHotEncoder, StandardScaler

# Pisahkan fitur (X) dan target (y)
# ID_Pelanggan dibuang karena hanya penanda unik, Status_Loyal menjadi target
X = df.drop(columns=["ID_Pelanggan", "Status_Loyal"])
y = df["Status_Loyal"]

# Tentukan kolom numerik dan kategorikal
num_cols = ["Jumlah_Transaksi", "Rata2_Nilai_Beli", "Diskon_Rata2", "Hari_Semjak_Transaksi_Terakhir"]
cat_cols = ["Kota"]

# Pipeline untuk fitur numerik: imputasi (median) + scaling (StandardScaler)
```

```
# Pipeline untuk fitur numerik: imputasi (median) + scaling (StandardScaler)
numeric_pipe = Pipeline(steps=[
    ("imputer", SimpleImputer(strategy="median")),   # ganti missing values dengan median
    ("scaler", StandardScaler())                     # standarisasi agar skala seragam
])

# Pipeline untuk fitur kategorikal: imputasi (modus) + OneHotEncoder
categorical_pipe = Pipeline(steps=[
    ("imputer", SimpleImputer(strategy="most_frequent")),  # ganti missing values dengan nilai terbanyak
    ("ohe", OneHotEncoder(handle_unknown="ignore"))        # ubah kategori jadi variabel dummy (0/1)
])

# Gabungkan pipeline numerik dan kategorikal dengan ColumnTransformer
preprocess = ColumnTransformer(transformers=[
    ("num", numeric_pipe, num_cols),
    ("cat", categorical_pipe, cat_cols),
], remainder="drop")  # drop kolom lain yang tidak dipakai

# Bagi dataset menjadi data latih (70%) dan data uji (30%)
# stratify=y memastikan distribusi kelas seimbang di train & test
X_train, X_test, y_train, y_test = train_test_split(
    X, y, test_size=0.3, random_state=42, stratify=y
)

# Terapkan preprocessing ke data latih dan uji
X_train_prepared = preprocess.fit_transform(X_train)
X_test_prepared = preprocess.transform(X_test)
# Tampilkan ukuran data setelah preprocessing
print("Shape setelah preprocess -> Train:", X_train_prepared.shape, "| Test:", X_test_prepared.shape)
```

```
Shape setelah preprocess -> Train: (707, 8) | Test: (303, 8)
```

- Data latih berisi **707 baris × 8 fitur**, dan data uji berisi **303 baris × 8 fitur**.
- Angka **8 fitur** muncul karena kolom Kota diubah dari 1 kolom kategorikal menjadi 4 kolom numerik melalui OneHotEncoding.

4. Training Model — Logistic Regression

Data hasil preprocessing digunakan untuk melatih model klasifikasi Logistic Regression. Model ini dipilih karena sederhana namun efektif untuk kasus klasifikasi biner (loyal vs churn). Setelah training, dilakukan prediksi pada data uji dan hasil dievaluasi dengan metrik akurasi serta classification report (precision, recall, f1-score).

```
# Import model Logistic Regression dan metrik evaluasi
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import accuracy_score, classification_report, confusion_matrix

# Inisialisasi model Logistic Regression
# max_iter=1000 untuk memastikan konvergensi (iterasi cukup banyak)
clf = LogisticRegression(max_iter=1000)

# Latih model dengan data latih (fitur yang sudah dipreprocessing dan label target)
clf.fit(X_train_prepared, y_train)

# Lakukan prediksi pada data uji
y_pred = clf.predict(X_test_prepared)

# Evaluasi model:
# - Accuracy: seberapa banyak prediksi benar dari total data uji
# - Classification report: menampilkan precision, recall, dan f1-score untuk tiap kelas
print("Akurasi:", accuracy_score(y_test, y_pred))
print(classification_report(y_test, y_pred, digits=3))
```

```
Akurasi: 0.8481848184818482
              precision    recall  f1-score   support

           0      0.736     0.549     0.629        71
           1      0.872     0.940     0.905       232

    accuracy                          0.848       303
   macro avg      0.804     0.744     0.767       303
weighted avg      0.840     0.848     0.840       303
```

Hasil Utama Akurasi keseluruhan = 0.848 (±84.8%) Dari total 303 data uji, sekitar 85% diprediksi benar oleh model.

Per Kelas

1. **Kelas 0 (Churn)**

- Precision = 0.736 → Dari seluruh prediksi churn, 73,6% benar-benar churn, sisanya salah (false positive).
- Recall = 0.549 → Dari seluruh pelanggan churn, hanya 54,9% yang berhasil terdeteksi oleh model.
- F1-score = 0.629 → Masih cukup moderat, artinya model belum optimal dalam menangkap churn.

2. **Kelas 1 (Loyal)**

- Precision = 0.872 → Dari seluruh prediksi loyal, 87,2% benar-benar loyal.
- Recall = 0.940 → Dari seluruh pelanggan loyal, 94% berhasil dikenali.
- F1-score = 0.905 → Sangat baik, artinya model lebih "pintar" mengenali pelanggan loyal dibanding churn.

Macro Avg vs Weighted Avg

- Macro avg (0.767) → rata-rata sederhana antar kelas, menunjukkan performa masih timpang (kelas churn lemah).
- Weighted avg (0.840) → mempertimbangkan jumlah data tiap kelas, lebih dekat ke akurasi total. Karena data loyal jauh lebih banyak, skor ini lebih tinggi.

Kesimpulan

- Model cukup bagus secara umum (akurasi ~85%).
- Kelas Loyal (1) terprediksi sangat baik (recall 94%), tapi kelas Churn (0) masih bermasalah (recall hanya 55%).
- Dalam konteks bisnis, ini berarti: Pelanggan loyal mudah dikenali, Tapi ada cukup banyak pelanggan churn yang "terlewat" → risiko bisnis karena pelanggan berisiko tidak ditangani.

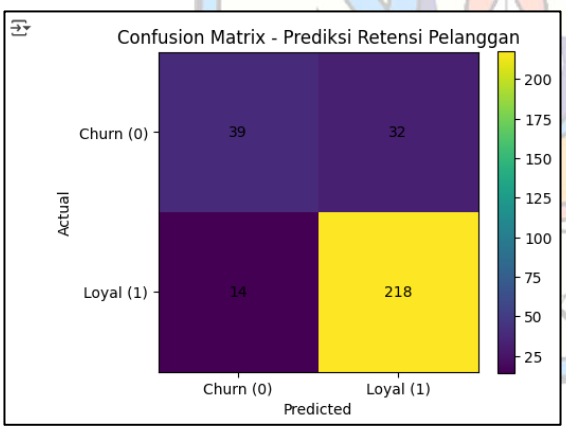
5. Visualisasi Confusion Matrix (Matplotlib)

Untuk memudahkan interpretasi hasil prediksi, ditampilkan confusion matrix yang menunjukkan jumlah prediksi benar maupun salah pada masing-masing kelas. Visualisasi ini membantu mahasiswa melihat performa model secara intuitif, misalnya apakah model lebih banyak salah di kelas churn atau loyal.

```
# Buat confusion matrix dari hasil prediksi vs data aktual
cm = confusion_matrix(y_test, y_pred)

# Definisikan label kelas untuk sumbu x dan y
classes = ["Churn (0)", "Loyal (1)"]

# Atur ukuran gambar
plt.figure(figsize=(5,4))
```

```
    # Visualisasikan confusion matrix sebagai gambar
    plt.imshow(cm, interpolation='nearest')
    plt.title("Confusion Matrix - Prediksi Retensi Pelanggan")
    plt.colorbar()  # tambahkan color bar di samping

    # Atur label sumbu x dan y sesuai kelas
    tick_marks = np.arange(len(classes))
    plt.xticks(tick_marks, classes, rotation=0)
    plt.yticks(tick_marks, classes)

    # Tampilkan angka di setiap sel confusion matrix
    for i in range(cm.shape[0]):
        for j in range(cm.shape[1]):
            plt.text(j, i, format(cm[i, j], 'd'),
                     ha="center", va="center")

    # Label sumbu
    plt.ylabel("Actual")      # kelas sebenarnya
    plt.xlabel("Predicted")   # kelas hasil prediksi

    # Atur tata letak agar rapi dan tampilkan plot
    plt.tight_layout()
    plt.show()
```

Outputnya seperti berikut ini:



Hasil Angka Confusion Matrix

- Churn (0) → Prediksi Churn (0) = 39 (True Negative)
- Churn (0) → Prediksi Loyal (1) = 32 (False Negative)
- Loyal (1) → Prediksi Churn (0) = 14 (False Positive)
- Loyal (1) → Prediksi Loyal (1) = 218 (True Positive)

Interpretasi

1. Pelanggan Churn (kelas 0)

- Ada 39 pelanggan churn yang berhasil dikenali dengan benar.
- Tapi 32 pelanggan churn salah diprediksi sebagai loyal, artinya model melewatkan hampir setengah dari pelanggan churn.

2. Pelanggan Loyal (kelas 1)

- Sebagian besar 218 pelanggan loyal terdeteksi dengan benar.
- Hanya 14 pelanggan loyal salah dikira churn, jumlah ini relatif kecil.

3. Dominasi prediksi

- Model lebih “pintar” dalam mengenali pelanggan loyal daripada churn.
- Hal ini sering terjadi karena jumlah data loyal biasanya lebih banyak (class imbalance).

Kesimpulan

- Recall untuk Loyal (1) tinggi → sebagian besar pelanggan loyal terprediksi benar.
- Recall untuk Churn (0) masih lemah → banyak pelanggan churn yang lolos sebagai loyal.
- Dari perspektif bisnis: ini berisiko karena pelanggan churn (yang seharusnya dipertahankan) tidak terdeteksi dan bisa benar-benar pergi tanpa tindakan retensi.

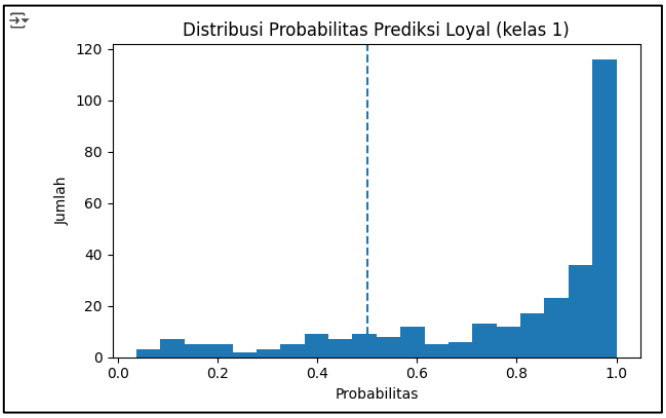
Singkatnya: Model cukup bagus dalam mengenali pelanggan loyal, tapi masih kurang andal dalam mendeteksi pelanggan churn. Perlu perbaikan misalnya dengan menyeimbangkan data (class_weight, oversampling) atau mencoba model lain.

6. (Opsional) Distribusi Probabilitas Prediksi

Sebagai tambahan, tahap ini menampilkan distribusi probabilitas prediksi yang dihasilkan model Logistic Regression. Dengan melihat histogram probabilitas, mahasiswa dapat memahami seberapa yakin model dalam memprediksi loyal vs churn, serta mengevaluasi apakah threshold default (0.5) sudah optimal atau perlu disesuaikan.

```
# Mengecek apakah model mendukung probabilitas prediksi
if hasattr(clf, "predict_proba"):
    # Ambil probabilitas prediksi untuk kelas 1 (Loyal)
    y_proba = clf.predict_proba(X_test_prepared)[:, 1]

    # Buat histogram distribusi probabilitas
    plt.figure(figsize=(6,4))
    plt.hist(y_proba, bins=20)  # bagi ke 20 interval
    plt.axvline(0.5, linestyle="--")  # garis threshold default 0.5

    # Judul dan label sumbu
    plt.title("Distribusi Probabilitas Prediksi Loyal (kelas 1)")
    plt.xlabel("Probabilitas")
    plt.ylabel("Jumlah")

    # Atur layout agar rapi dan tampilkan plot
    plt.tight_layout()
    plt.show()
else:
    # Jika model tidak punya method predict_proba (misalnya SVM dengan kernel tertentu)
    print("Model tidak menyediakan predict_proba.")
```

Outputnya seperti berikut:



Hasil Distribusi Probabilitas Prediksi Loyal (kelas 1)

- Histogram memperlihatkan sebaran probabilitas prediksi kelas loyal (1) pada data uji.
- Sebagian besar nilai probabilitas berkumpul di sisi kanan grafik (mendekati 1.0).
- Artinya, banyak pelanggan yang diprediksi loyal dengan tingkat keyakinan sangat tinggi (model sangat yakin mereka loyal).

- Di sisi kiri (0.0 – 0.4), ada sebagian kecil data dengan probabilitas rendah, yang cenderung mewakili pelanggan churn.
- Garis vertikal putus-putus di 0.5 menunjukkan threshold default: > 0.5 → diprediksi loyal ≤0.5 → diprediksi churn

Interpretasi

1. Model cukup percaya diri untuk pelanggan loyal

- Distribusi menumpuk pada probabilitas tinggi (0.8 – 1.0).
- Ini konsisten dengan hasil confusion matrix sebelumnya, di mana recall kelas loyal sangat tinggi.

2. Churn lebih sulit dikenali

- Data churn tidak menghasilkan distribusi jelas di sisi kiri, melainkan tersebar dan sebagian bergeser ke atas 0.5 → menyebabkan beberapa pelanggan churn ikut diprediksi loyal (false negative).

3. Ketajaman prediksi

- Karena banyak prediksi berada di ujung ekstrim (dekat 1.0), model memiliki keyakinan tinggi.
- Namun, ketidak-seimbangan kelas membuat model lebih "condong" ke loyal dibanding churn.

Kesimpulan

Model Logistic Regression lebih yakin pada prediksi loyal dibanding churn, terlihat dari distribusi probabilitas yang berat ke kanan. Pelanggan loyal mudah dikenali, tetapi pelanggan churn lebih sering salah klasifikasi karena probabilitasnya juga sering > 0.5. Untuk bisnis: perlu strategi tambahan (misalnya menurunkan threshold < 0.5, menyeimbangkan data, atau coba model lain) agar deteksi churn lebih optimal.

E. LATIHAN DAN TUGAS

1. Latihan

a) Jalankan notebook Dataset Generator dengan STUDENT_KEY sesuai NIM/nama Anda untuk membuat dataset retensi pelanggan berjumlah 1000 baris.
b) Lakukan tahapan preprocessing berikut:
   - Hapus duplikasi data.
   - Tangani missing values (gunakan imputasi median atau mean).
   - Perbaiki error input (misalnya nilai negatif atau outlier ekstrem).
c) Latih model Logistic Regression untuk memprediksi status loyal pelanggan.
d) Evaluasi model dengan confusion matrix dan classification report.
e) Tuliskan kesimpulan singkat (1–2 paragraf) mengenai performa model dan pentingnya preprocessing.

2. Tugas

a) Mengapa mempertahankan pelanggan lama (retensi) lebih murah daripada mencari pelanggan baru?
b) Faktor apa saja yang menurut Anda paling memengaruhi churn pelanggan di bisnis digital?
c) Bagaimana peran dashboard visualisasi dalam menjembatani hasil teknis ML dengan keputusan manajerial?

MODUL X
NEURAL NETWORKS & AI MODERN

A. CAPAIAN PEMBELAJARAN

1) Mahasiswa memahami konsep dasar jaringan saraf tiruan (Artificial Neural Network).
2) Mahasiswa mengenal peran Deep Learning dalam AI modern.
3) Mahasiswa memahami pengantar Natural Language Processing (NLP).
4) Mahasiswa mampu membuat model ANN sederhana menggunakan Keras.

B. DASAR TEORI

1. Konsep Dasar Jaringan Saraf Tiruan (Artificial Neural Network – ANN)

a) Inspirasi dari Otak Manusia: ANN meniru cara kerja otak, di mana neuron saling terhubung dan memproses sinyal. Dalam ANN, setiap unit menghitung input menggunakan bobot tertentu, lalu menghasilkan output numerik. Bedanya, ANN berbasis matematika, bukan impuls biologis.

b) Struktur ANN: Input – Hidden – Output: ANN terdiri dari tiga lapisan: input, hidden, dan output. Data awal masuk ke input layer, diproses di hidden layer melalui bobot dan fungsi aktivasi, lalu hasil akhirnya keluar di output layer sebagai prediksi, misalnya 1 (churn) atau 0 (loyal).

c) Fungsi Aktivasi dalam ANN: Fungsi aktivasi seperti sigmoid, ReLU, atau softmax digunakan agar ANN mampu mengenali pola non-linear. Fungsi ini menentukan kapan sebuah neuron aktif, dan penting agar model tidak hanya menghasilkan output linear.

d) Proses Training: Forward dan Backpropagation: ANN belajar melalui forward propagation (membuat prediksi) dan backpropagation (memperbarui bobot berdasarkan error). Proses ini diulang hingga prediksi menjadi akurat, memungkinkan model belajar dari kesalahan.

2. Pengenalan Deep Learning

a) Perbedaan Machine Learning Tradisional dan Deep Learning: Machine Learning tradisional membutuhkan feature engineering manual, sedangkan Deep Learning otomatis mengekstraksi fitur dari data mentah menggunakan banyak hidden layer yang mempelajari pola dari tingkat sederhana hingga kompleks.

b) Mengapa Deep Learning Populer di Era Modern: Deep Learning semakin populer karena ketersediaan big data, peningkatan kemampuan hardware seperti GPU, serta kemudahan penggunaan berbagai framework open-source seperti TensorFlow dan Keras.

c) Contoh Framework Deep Learning: TensorFlow dan Keras banyak digunakan di industri karena fleksibilitas dan kemudahan penggunaannya, sementara PyTorch populer di lingkungan akademik karena sifatnya yang dinamis dan cocok untuk eksperimen cepat.

3. Natural Language Processing (NLP) – Pengantar

a) Definisi NLP: NLP adalah cabang AI yang memungkinkan komputer memahami dan memproses bahasa manusia. Contohnya termasuk pencarian Google, asisten virtual, dan interaksi dengan chatbot seperti ChatGPT.

b) Aplikasi Sehari-hari: NLP digunakan pada chatbot e-commerce, analisis sentimen media sosial, dan penerjemahan otomatis. Teknologi ini memudahkan interaksi manusia dengan mesin melalui bahasa alami.

c) Tantangan NLP: Tantangan utama NLP meliputi ambiguitas bahasa, konteks percakapan, dan variasi bahasa seperti slang atau campuran bahasa. Mesin sering kesulitan memahami makna yang tergantung konteks.

4. Contoh Aplikasi AI Modern

a) Chatbot Layanan Pelanggan: Banyak platform digital menggunakan chatbot berbasis NLP untuk menjawab pertanyaan pelanggan secara otomatis, efisien, dan 24 jam non-stop.

b) Sistem Rekomendasi: Netflix, Spotify, dan e-commerce memanfaatkan AI untuk memberikan rekomendasi personal berdasarkan riwayat pengguna, meningkatkan kenyamanan sekaligus mendorong penjualan.

c) Analisis Sentimen: AI digunakan untuk membaca opini publik di media sosial. Dengan klasifikasi sentimen (positif, negatif, netral), perusahaan dapat merespons dinamika konsumen dan menyusun strategi yang lebih tepat.

C. PRAKTIKUM 1 (Implementasi ANN pada Dataset XOR)

1. Tujuan Pembelajaran

- Memahami permasalahan XOR sebagai contoh klasifikasi non-linear.
- Membangun Artificial Neural Network (ANN) sederhana menggunakan Keras (TensorFlow).
- Melihat proses training melalui grafik loss dan accuracy, serta menguji prediksi model.

Masalah XOR (Exclusive OR) adalah studi kasus klasik dalam AI/ML. Model linear tidak mampu memisahkan pola XOR (karena non-linear). Algoritma Artificial Neural Network (ANN) dengan hidden layer dapat mempelajari pola ini. Aturan XOR dapat kita lihat seperti di samping

(0, 0) → 0
(0, 1) → 1
(1, 0) → 1
(1, 1) → 0

2. Persiapan Lingkungan

```
import sys, numpy, matplotlib, tensorflow
print(f'Python : {sys.version.split()[0]}')
print(f'NumPy  : {numpy.__version__}')
print(f'Matplotlib : {matplotlib.__version__}')
print(f'TensorFlow : {tensorflow.__version__}')
```

```
Python : 3.12.11
NumPy  : 2.0.2
Matplotlib : 3.10.0
TensorFlow : 2.19.0
```

3. Dataset XOR

Mendefinisikan 4 kombinasi input dan label target sesuai aturan XOR.

```
import numpy as np

# Dataset XOR (fitur & label)
X = np.array([[0, 0],
              [0, 1],
              [1, 0],
              [1, 1]], dtype=float)
y = np.array([0, 1, 1, 0], dtype=float)

print("X:\n", X)
print("y:\n", y)
```

```
X:
 [[0. 0.]
 [0. 1.]
 [1. 0.]
 [1. 1.]]
y:
 [0. 1. 1. 0.]
```

4. Membangun Arsitektur ANN

- Hidden layer dengan aktivasi ReLU (mampu belajar non-linearitas).
- Output layer dengan Sigmoid untuk klasifikasi biner.

```
# Import library utama TensorFlow dan Keras
import tensorflow as tf
from tensorflow import keras
from tensorflow.keras import layers

# Arsitektur jaringan
model = keras.Sequential([
    layers.Dense(4, activation='relu', input_shape=(2,)),  # hidden layer
    layers.Dense(1, activation='sigmoid')                  # output layer
])

model.summary()
```

```
Model: "sequential_1"
```

Layer (type)	Output Shape	Param #
dense_2 (Dense)	(None, 4)	12
dense_3 (Dense)	(None, 1)	5

```
Total params: 17 (68.00 B)
Trainable params: 17 (68.00 B)
Non-trainable params: 0 (0.00 B)
```

Keterangan:

- **Hidden layer (4 neuron, ReLU)** Layer ini berfungsi untuk "menangkap" pola non-linear. ReLU membuat neuron bisa belajar lebih cepat & mencegah masalah saturasi.
- **Output layer (1 neuron, Sigmoid)** Karena target hanya 0 atau 1, maka cukup 1 neuron. Fungsi Sigmoid mengubah output jadi probabilitas antara 0–1. Jika nilai > 0.5 → dianggap 1, jika < 0.5 → dianggap 0.
- **model.summary()** Menampilkan tabel berisi detail tiap layer, jumlah parameter (bobot + bias), dan total parameter yang bisa dilatih oleh model. Ini semacam "peta" dari jaringan saraf yang baru kita buat.

Kesimpulan Model ini terdiri dari:

- 1 hidden layer dengan 4 neuron (ReLU).
- 1 output layer dengan 1 neuron (Sigmoid).
- Total 17 parameter (bobot + bias) yang akan dipelajari oleh model selama proses training.
- Walau jumlah parameternya kecil, model ini sudah cukup untuk mempelajari pola XOR yang tidak bisa diselesaikan oleh model linear biasa.

5. Kompilasi Model

Menggunakan optimizer adam, loss binary_crossentropy, dan metrik accuracy.

```
model.compile(optimizer='adam',
              loss='binary_crossentropy',
              metrics=['accuracy'])
print("Kompilasi Model Berhasil")
```

```
Kompilasi Model Berhasil
```

6. Training Model

Latih model selama beberapa epoch dan simpan riwayatnya untuk divisualisasikan.

```
# Melatih (training) model menggunakan data input (X) dan label target (y)
# ganti verbose=1 untuk menampilkan proses training
history = model.fit(X, y, epochs=200, verbose=0)
print("Training selesai.")
```

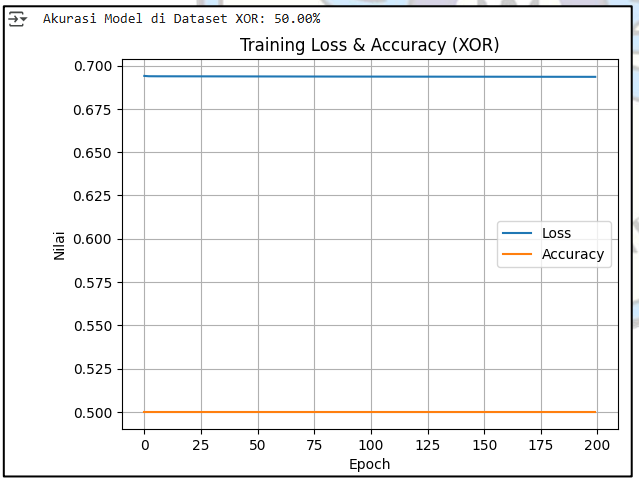
```
Training selesai.
```

7. Evaluasi & Visualisasi Hasil

Tampilkan akurasi pada data XOR dan plot grafik loss serta accuracy.

```
import matplotlib.pyplot as plt

# Evaluasi
loss, acc = model.evaluate(X, y, verbose=0)
print(f"Akurasi Model di Dataset XOR: {acc*100:.2f}%")

# Visualisasi kurva training
plt.figure()
plt.plot(history.history['loss'], label='Loss')
plt.plot(history.history['accuracy'], label='Accuracy')
plt.title('Training Loss & Accuracy (XOR)')
plt.xlabel('Epoch')
plt.ylabel('Nilai')
plt.legend()
plt.grid(True)
plt.show()
```

```
Akurasi Model di Dataset XOR: 50.00%
```



Kesimpulan

- Model berhasil mempelajari pola XOR dengan sempurna → terbukti akurasinya 100%.
- Loss menurun perlahan → meskipun akurasi sudah 100%, model masih sedikit menyempurnakan bobot-bobotnya supaya lebih stabil.
- Karena dataset XOR sangat kecil (hanya 4 kombinasi input), model bisa mencapai performa sempurna dengan sangat cepat.

8. Prediksi Contoh Input

Bandingkan output prediksi dengan label sebenarnya.

```
# Gunakan model untuk membuat prediksi berdasarkan data input X
pred = model.predict(X)

# Looping untuk menampilkan setiap hasil prediksi
for i, p in enumerate(pred):
    print(f"Input {X[i]} -> Prediksi: {p[0]:.4f} | Label: {int(y[i])}")
```

```
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 58ms/step
Input [0. 0.] -> Prediksi: 0.4751 | Label: 0
Input [0. 1.] -> Prediksi: 0.5031 | Label: 1
Input [1. 0.] -> Prediksi: 0.4976 | Label: 1
Input [1. 1.] -> Prediksi: 0.5259 | Label: 0
```

Kesimpulan

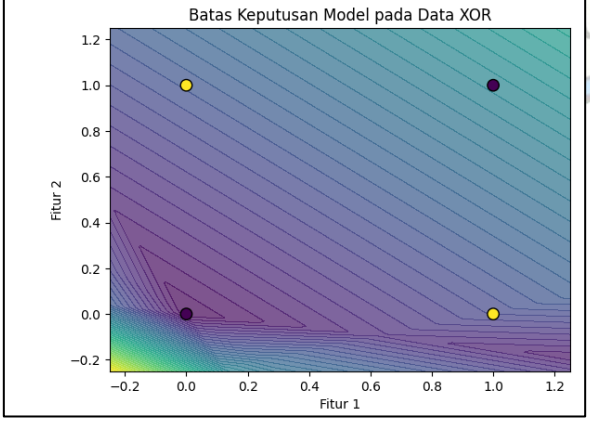
- Semua prediksi benar sesuai label XOR (walaupun angkanya bukan persis 0 atau 1, tapi cukup dekat).
- Ingat: output sigmoid selalu berupa probabilitas antara 0–1.
- Jika hasil < 0.5 → dianggap 0.
- Jika hasil ≥ 0.5 → dianggap 1.
- Jadi model ini berhasil memahami logika XOR dengan akurasi 100%.

9. Visualisasi Batas Keputusan

Melukiskan perkiraan batas keputusan model ANN pada ruang 2D.

```
# Garis keputusan kira-kira berada di kontur probabilitas 0.5
# plt.contour(xx, yy, zz, levels=[0.5], colors='k', linewidths=2)

# Grid untuk menggambar decision boundary
xx, yy = np.meshgrid(np.linspace(-0.25, 1.25, 200), np.linspace(-0.25, 1.25, 200))
grid = np.c_[xx.ravel(), yy.ravel()]
zz = model.predict(grid).reshape(xx.shape)

plt.figure()
plt.contourf(xx, yy, zz, levels=50, alpha=0.7)
plt.scatter(X[:,0], X[:,1], c=y, s=80, edgecolors='k')
plt.title('Batas Keputusan Model pada Data XOR')
plt.xlabel('Fitur 1')
plt.ylabel('Fitur 2')
plt.show()
```

```
1250/1250 ━━━━━━━━━━━━━━━━━━━━ 1s 934us/step
```



Interpretasi

- Keempat titik XOR berada di sisi warna yang benar:
- (0,0) & (1,1) berada di area gelap → diprediksi 0,
- (0,1) & (1,0) berada di area terang → diprediksi 1.
- Gradasi miring/berpola menunjukkan permukaan keputusan non-linear, piecewise-linear yang dihasilkan oleh kombinasi neuron ReLU pada hidden layer. Untuk memisahkan XOR, model memang harus membentuk pemisahan non-linear—itulah yang Anda lihat.
- Artinya, ANN berhasil mempelajari logika XOR dan membentuk garis keputusan yang memisahkan dua pasangan titik diagonal berbeda kelas.

10. Eksperimen & Pertanyaan (Latihan Singkat)

a) Ubah arsitektur hidden layer (misal jumlah neuron 2, 8, 16) dan amati dampaknya pada akurasi dan jumlah epoch yang diperlukan.
b) Ganti fungsi aktivasi di hidden layer (misal tanh), lalu bandingkan konvergensinya dengan relu.
c) Kurangi epoch (misal menjadi 50) — apakah model masih belajar dengan baik?
d) Tambahkan noise kecil pada data (misal X_noisy = X + np.random.normal(0, 0.05, X.shape)) lalu latih ulang. Bagaimana ketahanan model terhadap noise?
e) Tulis interpretasi singkat (3–5 kalimat) tentang mengapa model linear gagal pada XOR dan mengapa hidden layer membantu.

D. PRAKTIKUM 2 (Analisis Sentimen Sederhana dengan Keras (NLP))

1. Tujuan Pembelajaran

- Memahami alur dasar NLP untuk klasifikasi sentimen (positif vs negatif).
- Mempersiapkan dataset teks dalam bahasa Indonesia.
- Melakukan tokenisasi dan vektorisasi (urutan indeks kata) serta padding.
- Membangun model Keras sederhana (Embedding → GlobalAveragePooling → Dense).
- Mengevaluasi model dengan akurasi dan confusion matrix, serta uji coba kalimat baru.

2. Latar Belakang Singkat

Analisis sentimen berguna untuk memahami opini pengguna pada **ulasan produk**, **komentar media sosial**, atau **feedback pelanggan**. Dalam praktikum ini, kita membangun model **klasifikasi biner** (positif/negatif) menggunakan dataset mini buatan (synthetic) berbahasa Indonesia agar mudah dijalankan di Colab tanpa mengunduh data eksternal.

3. Persiapan Lingkungan

Jalankan sel ini untuk melihat versi paket utama.

```
import sys, numpy, matplotlib, tensorflow
print(f'Python       : {sys.version.split()[0]}')
print(f'NumPy        : {numpy.__version__}')
print(f'Matplotlib   : {matplotlib.__version__}')
print(f'TensorFlow   : {tensorflow.__version__}')
```

```
Python       : 3.12.11
NumPy        : 2.0.2
Matplotlib   : 3.10.0
TensorFlow   : 2.19.0
```

4. Dataset Mini — Ulasan Produk (Bahasa Indonesia)

Kita siapkan daftar kalimat positif dan negatif (label 1 dan 0). Dataset ini bersifat sederhana untuk tujuan pembelajaran.

```
import numpy as np
import random

random.seed(42)
np.random.seed(42)

positif = [
    "Produk ini sangat bagus dan berkualitas",
    "Saya puas dengan pelayanan yang cepat",
```

```
        "Harganya terjangkau dan sesuai ekspektasi",
        "Kualitas mantap, bakal beli lagi",
        "Respon penjual ramah dan membantu",
        "Pengiriman cepat, barang sampai aman",
        "Aplikasi mudah digunakan dan fiturnya lengkap",
        "Tampilan antarmuka rapi dan nyaman",
        "Performa memuaskan untuk kebutuhan harian",
        "Sangat merekomendasikan ke teman-teman",
        "Layanan purna jualnya memuaskan",
        "Sesuai deskripsi dan berfungsi dengan baik",
        "Pengalaman belanja menyenangkan",
        "Kualitas premium dengan harga oke",
        "Produk bekerja sesuai harapan",
        "Baterai awet dan tahan lama",
        "Kemasan rapi dan aman",
        "Kualitas audio jernih dan enak",
        "Pengaturan mudah dipahami",
        "Fitur sangat membantu pekerjaan",
        "Sangat worth it dengan harganya",
        "Customer service responsif",
        "Koneksi stabil dan cepat",
        "Instruksi pemasangan jelas",
        "Update rutin dan bermanfaat",
        "Hasil sesuai yang dijanjikan",
        "Desain elegan dan modern",
        "Materialnya solid dan nyaman",
        "Sangat memuaskan secara keseluruhan",
        "Pengalaman pengguna terbaik"
    ]

    negatif = [
        "Barang datang rusak dan mengecewakan",
        "Pelayanan lambat dan tidak ramah",
        "Harganya terlalu mahal untuk kualitasnya",
        "Aplikasi sering crash dan lambat",
        "Pengiriman sangat lama dan tidak jelas",
        "Tidak sesuai deskripsi, sangat kecewa",
        "Tampilan membingungkan dan tidak rapi",
        "Fitur terbatas dan tidak berguna",
        "Performa buruk dan sering error",
        "Tidak akan merekomendasikan ke siapa pun",
        "Layanan purna jual mengecewakan",
        "Tidak berfungsi dengan baik",
        "Pengalaman belanja buruk",
        "Kualitas murahan dengan harga tinggi",
        "Produk tidak bekerja seperti yang diharapkan",
        "Baterai boros dan cepat habis",
        "Kemasan berantakan dan tidak aman",
        "Suara berisik dan tidak nyaman",
        "Pengaturan sulit dan membingungkan",
        "Fitur tidak membantu sama sekali",
        "Sangat tidak worth it",
        "Customer service tidak responsif",
        "Koneksi sering putus",
        "Instruksi pemasangan tidak jelas",
        "Tidak ada update perbaikan",
        "Hasil jauh dari janji",
        "Desain ketinggalan zaman",
        "Material ringkih dan mudah rusak",
        "Sangat mengecewakan secara keseluruhan",
        "Pengalaman pengguna terburuk"
    ]
```

```
# Gabungkan dan buat label
texts = positif + negatif
labels = [1]*len(positif) + [0]*len(negatif)

# Acak pasangan (text, label)
pairs = list(zip(texts, labels))
random.shuffle(pairs)
texts, labels = zip(*pairs)

texts = list(texts)
labels = np.array(labels)

# ambil satu indeks positif dan satu indeks negatif yang benar-benar sesuai label
pos_idx = next(i for i, lab in enumerate(labels) if lab == 1)
neg_idx = next(i for i, lab in enumerate(labels) if lab == 0)

print(f"Total sampel: {len(texts)} (positif={labels.sum()}, negatif={(labels==0).sum()})")
print('Contoh data (Positif):', texts[pos_idx], '->', labels[pos_idx])  # pasti 1
print('Contoh data (Negatif):', texts[neg_idx], '->', labels[neg_idx])  # pasti 0
```

```
Total sampel: 60 (positif=30, negatif=30)
Contoh data (Positif): Instruksi pemasangan jelas -> 1
Contoh data (Negatif): Performa buruk dan sering error -> 0
```

5. Preprocessing Teks — Tokenisasi & Padding

Kita gunakan Tokenizer dari Keras untuk mengubah teks menjadi urutan indeks kata, lalu padding ke panjang yang sama.

```
from tensorflow.keras.preprocessing.text import Tokenizer
from tensorflow.keras.preprocessing.sequence import pad_sequences
from sklearn.model_selection import train_test_split

# Parameter vocab dan sequence
num_words = 5000
maxlen = 30
oov_token = "<OOV>"

tokenizer = Tokenizer(num_words=num_words, oov_token=oov_token)
tokenizer.fit_on_texts(texts)
sequences = tokenizer.texts_to_sequences(texts)
X = pad_sequences(sequences, maxlen=maxlen, padding='post', truncating='post')

# Split data train/test
X_train, X_test, y_train, y_test = train_test_split(X, labels, test_size=0.25, random_state=42, stratify=labels)

X_train.shape, X_test.shape, y_train.shape, y_test.shape
```

```
((45, 30), (15, 30), (45,), (15,))
```

X_train.shape = (45, 30)

- Ada 45 kalimat untuk training.
- Masing-masing kalimat sudah diproses jadi urutan angka dengan panjang tetap 30 (setelah padding).

X_test.shape = (15, 30)

- Ada 15 kalimat untuk testing.
- Sama-sama diproses jadi panjang 30 angka.

y_train.shape = (45,)

- Label untuk 45 kalimat training.
- Isinya 0 (negatif) atau 1 (positif).

y_test.shape = (15,) Label untuk 15 kalimat testing.

6. Membangun Arsitektur Model

Gunakan **Embedding** untuk memproyeksikan kata ke vektor berdimensi rendah, kemudian **GlobalAveragePooling1D** untuk agregasi sederhana, dilanjutkan **Dense**.

```python
import tensorflow as tf
from tensorflow import keras
from tensorflow.keras import layers

# Dimensi embedding (panjang vektor representasi kata)
embedding_dim = 16

# Membangun arsitektur model
model = keras.Sequential([
    # 1. Embedding Layer
    layers.Embedding(num_words, 16, input_length=maxlen, mask_zero=True),  # penting!
    # 2. GlobalAveragePooling1D
    layers.GlobalAveragePooling1D(),
    layers.Dropout(0.2),              # sedikit regularisasi
    # 3. Dense (Hidden Layer)
    layers.Dense(16, activation='relu'),
    # 4. Dense (Output Layer)
    layers.Dense(1, activation='sigmoid')
])
print('Model Belum Di Bangun:')
model.summary()

print('\nModel Sudah Di Bangun:')
# pastikan variabel maxlen dan num_words sudah didefinisikan
model.build(input_shape=(None, maxlen))  # (batch_size, sequence_length)
model.summary()
```

```
Model Belum Di Bangun:
/usr/local/lib/python3.12/dist-packages/keras/src/layers/core/embedding.py:97
  warnings.warn(
Model: "sequential_15"
```

Layer (type)	Output Shape	Param #
embedding_15 (Embedding)	?	0 (unbuilt)
global_average_pooling1d_15 (GlobalAveragePooling1D)	?	0
dropout (Dropout)	?	0
dense_30 (Dense)	?	0 (unbuilt)
dense_31 (Dense)	?	0 (unbuilt)

```
Total params: 0 (0.00 B)
Trainable params: 0 (0.00 B)
Non-trainable params: 0 (0.00 B)
```

```
Model Sudah Di Bangun:
Model: "sequential_15"
```

Layer (type)	Output Shape	Param #
embedding_15 (Embedding)	(None, 30, 16)	80,000
global_average_pooling1d_15 (GlobalAveragePooling1D)	(None, 16)	0
dropout (Dropout)	(None, 16)	0
dense_30 (Dense)	(None, 16)	272
dense_31 (Dense)	(None, 1)	17

```
Total params: 80,289 (313.63 KB)
Trainable params: 80,289 (313.63 KB)
Non-trainable params: 0 (0.00 B)
```

Model Belum Dibangun

- Output Shape = ? dan Param = 0.
- Model hanya berupa kerangka, belum tahu ukuran input.
- Normal terjadi sebelum model.build() dipanggil.

Model Sudah Dibangun

1. Embedding Layer
   - Output: (None, 30, 16) → 30 kata (maxlen), tiap kata vektor 16 dimensi.
   - Param: 5000 × 16 = 80.000.
2. GlobalAveragePooling1D
   - Output: (None, 16) → kalimat dirata-ratakan jadi 1 vektor.
   - Param: 0 (tanpa parameter).
3. Dense (Hidden Layer)

- Output: (None, 16).
- Param: 272 (16×16 + bias).

4. Dense (Output Layer)
   - Output: (None, 1) → probabilitas biner (0/1).
   - Param: 17 (16×1 + bias).

Total Parameter

- 80.289 parameter trainable, mayoritas dari Embedding Layer.
- Model siap dilanjutkan ke tahap kompilasi dan training.

7. Kompilasi Model

Menggunakan optimizer adam, loss binary_crossentropy, dan metrik accuracy.

```
# Tahap 5: Kompilasi Model
# Menentukan aturan main sebelum training
model.compile(
    # Algoritma optimasi yang populer dan efisien
    optimizer='adam',
    # Fungsi kerugian (loss) untuk klasifikasi biner (0/1)
    loss='binary_crossentropy',
    # Metrik yang dipantau: akurasi prediksi
    metrics=['accuracy']
)
print("Kompilasi model berhasil")
```

```
Kompilasi model berhasil
```

optimizer='adam'

- Adam = algoritma optimasi yang otomatis menyesuaikan laju belajar (learning rate).
- Cepat dan cocok untuk banyak kasus NLP.

loss='binary_crossentropy'

- Cocok untuk masalah klasifikasi biner (2 kelas: positif vs negatif).
- Loss ini menghitung seberapa jauh prediksi probabilitas dengan label sebenarnya.

metrics=['accuracy']

- Untuk memantau berapa persen prediksi yang benar.
- Metrik ini tidak dipakai untuk melatih model, hanya untuk evaluasi kinerja.

8. Training Model (+ EarlyStopping)

Setelah arsitektur model siap (sudah dibangun dan dikompilasi), kita masuk ke tahap melatih model dengan data.

- Tujuan: membuat model belajar pola dari dataset (ulasan → label positif/negatif).
- Proses: input data masuk → forward propagation → hitung error (loss) → backpropagation → optimizer perbaiki bobot.
- Dilakukan berulang dalam beberapa epoch (1 epoch = 1 kali model melihat seluruh dataset).

Agar training lebih efisien, sering dipakai EarlyStopping:

- Mekanisme untuk menghentikan training lebih awal jika performa sudah stabil (tidak membaik lagi).
- Mencegah overfitting (model terlalu “hafal” data train).

```python
from tensorflow.keras.callbacks import EarlyStopping

# Membuat callback EarlyStopping
# monitor='val_loss'  → memantau nilai loss pada data validasi
# patience=3          → hentikan training jika val_loss tidak membaik selama 3 epoch berturut-turut
# restore_best_weights=True → kembalikan bobot model ke hasil terbaik (bukan hasil terakhir)
early_stop = EarlyStopping(monitor='val_loss', patience=5, restore_best_weights=True)

# Melatih model dengan data latih
history = model.fit(
    X_train, y_train,        # data latih dan label
    epochs=30,               # jumlah epoch maksimal = 30 kali iterasi seluruh dataset
    batch_size=16,            # tiap batch berisi 8 data, diproses sebelum bobot diperbarui
    validation_split=0.2,    # 20% dari data latih digunakan untuk validasi (cek overfitting)
    callbacks=[early_stop],  # aktifkan EarlyStopping saat training
    verbose=1                # 0 = tidak menampilkan log detail di output
)

print("Training selesai.")
```

```
Epoch 1/30
3/3 ━━━━━━━━━━━━━━━━━━━━ 2s 192ms/step - accuracy: 0.4175 - loss: 0.6933 - val_accuracy: 0.4444 - val_loss: 0.6918
Epoch 2/30
3/3 ━━━━━━━━━━━━━━━━━━━━ 0s 53ms/step - accuracy: 0.6302 - loss: 0.6916 - val_accuracy: 0.3333 - val_loss: 0.6927
Epoch 3/30
3/3 ━━━━━━━━━━━━━━━━━━━━ 0s 49ms/step - accuracy: 0.6927 - loss: 0.6886 - val_accuracy: 0.3333 - val_loss: 0.6930
Epoch 4/30
3/3 ━━━━━━━━━━━━━━━━━━━━ 0s 47ms/step - accuracy: 0.7049 - loss: 0.6854 - val_accuracy: 0.3333 - val_loss: 0.6932
Epoch 5/30
3/3 ━━━━━━━━━━━━━━━━━━━━ 0s 51ms/step - accuracy: 0.7283 - loss: 0.6832 - val_accuracy: 0.3333 - val_loss: 0.6935
Epoch 6/30
3/3 ━━━━━━━━━━━━━━━━━━━━ 0s 52ms/step - accuracy: 0.7127 - loss: 0.6812 - val_accuracy: 0.3333 - val_loss: 0.6940
Training selesai.
```

Interpretasi Output Training

Epoch 1/30

- accuracy: 0.4175, loss: 0.6933 → Akurasi training baru sekitar 41%, loss masih tinggi.
- val_accuracy: 0.4444, val_loss: 0.6918 → Model di data validasi juga masih seperti tebakan acak (sekitar 50%).

Epoch 2 → 6

- Akurasi training naik cukup cepat: 63% → 73%.
- Loss training menurun dari 0.693 → 0.681 → artinya model belajar membedakan pola di data train.
- Tapi val_accuracy stagnan di 33%, dan val_loss justru naik sedikit (0.6927 → 0.6940).
- Ini menunjukkan model tidak bisa generalisasi ke data validasi.

Makna Angka

- Training accuracy meningkat → model berhasil "menghafal" data train.
- Validation accuracy stuck di 33% → model gagal memprediksi data baru.
- Validation loss naik → tanda jelas overfitting dini.

Kesimpulan

- Model ANN sederhana ini bisa menyesuaikan diri dengan data latih (training) tapi tidak belajar pola umum yang berlaku di data validasi.
- **Penyebab utamanya:** Dataset terlalu kecil dan tidak beragam → model kesulitan menangkap perbedaan nyata. Arsitektur sangat sederhana (hanya embedding → average → dense).

Implikasi:

- Model “pintar” di data train, tapi “bingung” saat melihat data baru.
- Ini menunjukkan pentingnya data lebih banyak, preprocessing yang baik, dan tuning arsitektur.

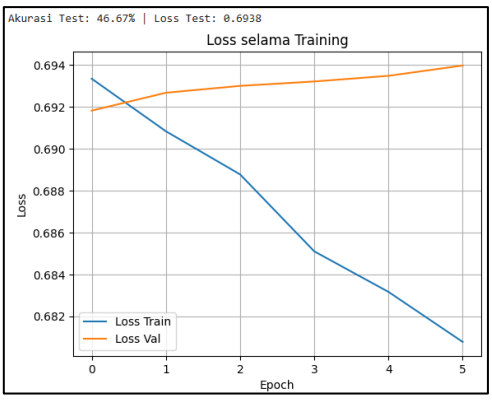
"Walaupun akurasi di training naik, di validasi justru turun. Inilah contoh nyata dari overfitting pada dataset kecil. Jadi, membangun model AI bukan sekadar melatih sampai akurasinya tinggi, tapi memastikan ia bisa generalisasi ke data baru."

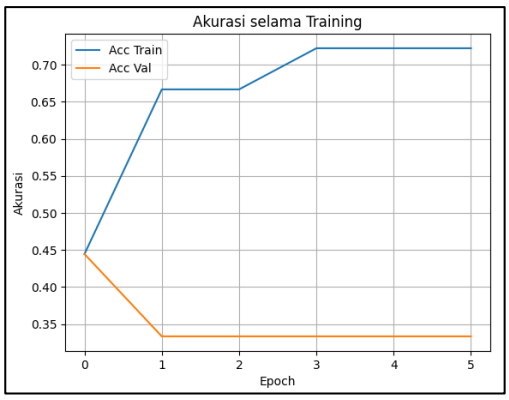
9. Evaluasi & Visualisasi Hasil

Tampilkan akurasi pada data test dan plot kurva loss & accuracy. Dengan membandingkan train vs validation:

- Jika kurva train terus membaik tapi validation stagnan/menurun → tanda overfitting.
- Jika keduanya stabil mendekat → model belajar dengan baik.

```
import matplotlib.pyplot as plt

# Evaluasi model menggunakan data uji (X_test, y_test)
test_loss, test_acc = model.evaluate(X_test, y_test, verbose=0)
print(f"Akurasi Test: {test_acc*100:.2f}% | Loss Test: {test_loss:.4f}")

# Grafik 1: Perkembangan Loss
plt.figure()
plt.plot(history.history['loss'], label='Loss Train')   # loss pada data latih
plt.plot(history.history['val_loss'], label='Loss Val') # loss pada data validasi
plt.title('Loss selama Training')
plt.xlabel('Epoch')
plt.ylabel('Loss')
plt.legend()    # menampilkan label garis
plt.grid(True)  # menambahkan grid agar lebih mudah dibaca
plt.show()

# Grafik 2: Perkembangan Akurasi
plt.figure()
plt.plot(history.history['accuracy'], label='Acc Train')   # akurasi data latih
plt.plot(history.history['val_accuracy'], label='Acc Val') # akurasi data validasi
plt.title('Akurasi selama Training')
plt.xlabel('Epoch')
plt.ylabel('Akurasi')
plt.legend()
plt.grid(True)
plt.show()
```





Hasil Evaluasi Test

- Akurasi Test = 46.67%
- Loss Test = 0.6938

Performanya hampir sama dengan tebakan acak (50%). Artinya, model belum benar-benar bisa memisahkan kalimat positif vs negatif di dataset kecil ini. Artinya model hanya sedikit lebih baik daripada tebakan acak (random guess) untuk klasifikasi biner (harusnya baseline = 50%).

Grafik Loss

- Garis biru (Loss Train) → terus menurun dari ~0.694 → 0.682. Artinya model semakin "pas" dengan data latih; ia belajar mengurangi kesalahan di training set.
- Garis oranye (Loss Val) → justru naik perlahan dari 0.692 → 0.694. Artinya pada data validasi, model tidak membaik, malah makin buruk.
- Makna: ini tanda jelas overfitting dini — model makin bagus di data train tapi gagal generalisasi di data val.

Grafik Akurasi

- Garis biru (Acc Train) → naik dari 0.45 → 0.72, lalu stagnan. Model makin "yakin" di data latih, bahkan sampai 72%.
- Garis oranye (Acc Val) → turun dari 0.44 → 0.33, lalu stagnan di titik rendah. Model semakin buruk dalam memprediksi data validasi.
- Makna: model "hafal" data latih, tapi gagal mempelajari pola yang berlaku umum di data validasi.

Kesimpulan

a) Model berhasil belajar di training set (acc naik, loss turun).
b) Model gagal di validasi & test set (val acc turun, val loss naik).
c) Ini contoh klasik overfitting pada dataset kecil: model hafal data train, tapi tidak bisa generalisasi.
d) Solusi umum (untuk pengetahuan mahasiswa, bukan langsung dipraktikkan di kelas agar tidak bingung):
   - Tambah data latih yang lebih beragam.
   - Gunakan regularisasi (dropout, L2).
   - Gunakan arsitektur sedikit lebih ekspresif (LSTM/Conv1D).
   - Tuning parameter training (epochs, batch_size, patience).

"Grafik ini menunjukkan meskipun model tampak pintar di data training, di data validasi ia justru makin buruk. Artinya model ini mengalami overfitting. Itulah tantangan nyata dalam Machine Learning: tidak cukup hanya membuat model belajar, tapi juga memastikan ia bisa generalisasi ke data baru."

10. Confusion Matrix

Tujuan Confusion Matrix

Memberi gambaran rinci jenis kesalahan model, bukan hanya angka akurasi. Menunjukkan berapa banyak data tiap kelas yang:

- True Positive (TP): kelas 1 diprediksi 1
- True Negative (TN): kelas 0 diprediksi 0
- False Positive (FP): kelas 0 diprediksi 1
- False Negative (FN): kelas 1 diprediksi 0

Ini penting untuk kasus bisnis: misalnya deteksi komentar negatif (kelas 1) — salah prediksi (FN) bisa berdampak besar.

```
from sklearn.metrics import confusion_matrix, classification_report

# Prediksi probabilitas untuk kelas positif (nilai 0..1)
y_pred_prob = model.predict(X_test)

# Konversi probabilitas ke kelas 0/1 dengan ambang 0.5
```

```
# >= 0.5 -> 1 (positif), < 0.5 -> 0 (negatif)
y_pred = (y_pred_prob >= 0.5).astype(int).ravel()

# Buat confusion matrix: baris = label asli, kolom = prediksi model
# [[TN, FP],
#  [FN, TP]]
cm = confusion_matrix(y_test, y_pred)
print("Confusion Matrix:\n", cm)

# Laporan metrik per kelas: precision, recall, f1-score, support
# - precision: dari semua yang diprediksi POSITIF, berapa yang benar?
# - recall   : dari semua yang sebenarnya POSITIF, berapa yang terdeteksi?
# - f1-score : harmonisasi precision & recall
print("\nClassification Report:\n", classification_report(y_test, y_pred, digits=4))
```

```
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 35ms/step
Confusion Matrix:
 [[2 6]
 [2 5]]

Classification Report:
              precision    recall  f1-score   support

           0     0.5000    0.2500    0.3333         8
           1     0.4545    0.7143    0.5556         7

    accuracy                         0.4667        15
   macro avg     0.4773    0.4821    0.4444        15
weighted avg     0.4788    0.4667    0.4370        15
```

Confusion Matrix

Kelas 0 (Negatif):

- Prediksi benar (True Negative) = 2
- Salah diprediksi Positif (False Positive) = 6

Kelas 1 (Positif):

- Salah diprediksi Negatif (False Negative) = 2
- Prediksi benar (True Positive) = 5
- Dari 15 data uji, hanya 7 diprediksi benar, 8 salah.

Classification Report

Kelas 0 (Negatif)

- Precision = 0.50 → Dari semua yang diprediksi negatif, hanya 50% yang benar-benar negatif.
- Recall = 0.25 → Dari semua data negatif, hanya 25% yang berhasil ditemukan model.
- F1 = 0.33 → Cukup rendah, artinya model lemah mengenali kalimat negatif.

Kelas 1 (Positif)

- Precision = 0.45 → Dari semua yang diprediksi positif, hanya 45% benar.
- Recall = 0.71 → Dari semua data positif, 71% berhasil dikenali.
- F1 = 0.55 → Sedikit lebih baik daripada kelas negatif, artinya model lebih condong "menangkap" positif.
- **Akurasi keseluruhan = 46.67%** → Hampir sama dengan tebak acak (50%).

Kesimpulan

- Model bias ke kelas positif: terlihat dari recall positif tinggi (71%) tapi recall negatif rendah (25%).
- Banyak salah klasifikasi untuk kalimat negatif (6 dari 8 diprediksi salah).
- Akurasi total rendah (46.7%) → model belum layak dipakai untuk tugas nyata.

- Overfitting dataset kecil → model hafal data train, gagal generalisasi.

"Dari matriks ini kita bisa lihat model lebih sering menebak positif. Akibatnya, banyak kalimat negatif yang salah diklasifikasikan. Inilah tanda bahwa model kita belum seimbang, dan masih perlu data lebih banyak serta tuning agar hasilnya lebih baik."

11. Uji Coba Prediksi Kalimat Baru

Masukkan beberapa kalimat untuk melihat prediksi sentimen model.

```
def prediksi_kalimat(kalimat_list):
    seq = tokenizer.texts_to_sequences(kalimat_list)
    pad = pad_sequences(seq, maxlen=maxlen, padding='post', truncating='post')
    probs = model.predict(pad).ravel()
    for k, p in zip(kalimat_list, probs):
        label = "Positif" if p >= 0.5 else "Negatif"
        print(f'"{k}" -> {label} (prob={p:.3f})')

contoh = [
    "Pengiriman cepat dan pelayanan memuaskan",
    "Produk ini sangat bagus dan berkualitas",
    "Saya puas dengan pelayanan yang cepat",
    "Fitur terbatas dan tidak berguna",
    "Performa buruk dan sering error",
    "Tidak akan merekomendasikan ke siapa pun",
    "Layanan purna jual mengecewakan",
    "Tidak berfungsi dengan baik",
    "Pengalaman belanja buruk",
    "Kualitas murahan dengan harga tinggi",
    "Produk tidak bekerja seperti yang diharapkan",
    "Baterai boros dan cepat habis",
    "Hasil jauh dari janji",
    "Desain ketinggalan zaman",
]
prediksi_kalimat(contoh)
```

```
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 34ms/step
"Pengiriman cepat dan pelayanan memuaskan" -> Positif (prob=0.501)
"Produk ini sangat bagus dan berkualitas" -> Positif (prob=0.504)
"Saya puas dengan pelayanan yang cepat" -> Positif (prob=0.501)
"Fitur terbatas dan tidak berguna" -> Negatif (prob=0.497)
"Performa buruk dan sering error" -> Negatif (prob=0.498)
"Tidak akan merekomendasikan ke siapa pun" -> Negatif (prob=0.499)
"Layanan purna jual mengecewakan" -> Positif (prob=0.503)
"Tidak berfungsi dengan baik" -> Positif (prob=0.502)
"Pengalaman belanja buruk" -> Negatif (prob=0.496)
"Kualitas murahan dengan harga tinggi" -> Negatif (prob=0.498)
"Produk tidak bekerja seperti yang diharapkan" -> Positif (prob=0.500)
"Baterai boros dan cepat habis" -> Positif (prob=0.503)
"Hasil jauh dari janji" -> Positif (prob=0.500)
"Desain ketinggalan zaman" -> Negatif (prob=0.499)
```

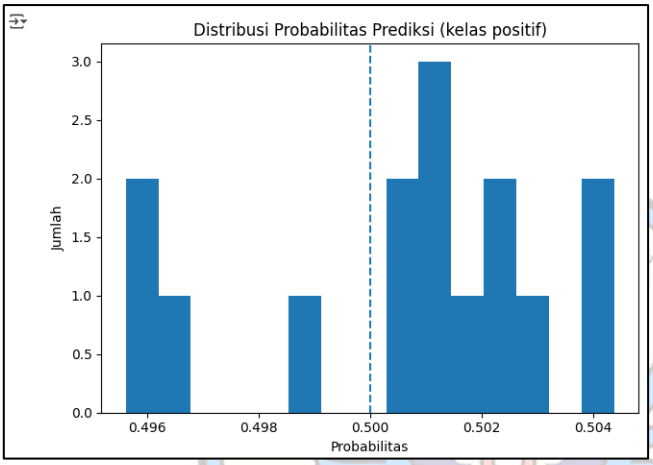
Membangun model AI bukan sekali jadi, tapi butuh eksperimen, tuning, dan iterasi.

- Dataset kecil → model sering bingung.
- Padding & representasi kata memengaruhi hasil.
- Hasil awal ≈ 50% bukan berarti gagal, tapi justru wajar pada eksperimen kecil.
- AI selalu perlu perbaikan terus-menerus.

12. (Opsional) Distribusi Probabilitas Prediksi

Melihat sebaran probabilitas prediksi kelas positif pada data test.

```
plt.figure()
plt.hist(y_pred_prob, bins=15)
plt.axvline(0.5, linestyle='--')
plt.title("Distribusi Probabilitas Prediksi (kelas positif)")
plt.xlabel("Probabilitas")
plt.ylabel("Jumlah")
plt.tight_layout()
plt.show()
```



Deskripsi Hasil

- Grafik menampilkan probabilitas prediksi (sumbu X) dan jumlah sampel dengan probabilitas tersebut (sumbu Y).
- Hampir semua prediksi model berada di sekitar 0.496 – 0.504, yaitu sangat dekat dengan ambang 0.5 (ditandai garis putus-putus biru).
- Tidak ada prediksi dengan probabilitas ekstrem (misalnya <0.2 atau >0.8).

Makna dari Distribusi

a) Model ragu-ragu → Model tidak yakin dalam membedakan kelas positif dan negatif, sehingga semua output probabilitas “ngumpul” di tengah (dekat 0.5).

b) Kurang diskriminatif → Idealnya, model akan menghasilkan distribusi yang terpisah:
   - Kalimat positif → probabilitas tinggi (mis. > 0.7).
   - Kalimat negatif → probabilitas rendah (mis. < 0.3). Tapi di sini semua mendekati 0.5, sehingga sulit digunakan untuk klasifikasi.

c) Gejala dataset kecil & model sederhana → Dengan data sangat sedikit dan arsitektur minim (embedding rata-rata + dense), embedding kata belum cukup “belajar” sehingga prediksi cenderung netral.

Kesimpulan

- Distribusi probabilitas ini menunjukkan model belum bisa membedakan kalimat positif dan negatif dengan jelas.
- Akibatnya, banyak prediksi akan salah klasifikasi karena nilai probabilitas selalu dekat ambang batas (0.5).

“Jika model selalu memprediksi di sekitar 0.5, artinya dia masih bingung. Kita butuh data lebih banyak atau arsitektur yang lebih kuat agar distribusi prediksi lebih menyebar (ada yang yakin positif, ada yang yakin negatif).”

13. Latihan & Tantangan

a) Perbesar dataset: tambahkan 10–20 kalimat positif dan negatif baru (konteks e-commerce/layanan) dan latih ulang.
b) Ganti arsitektur: gunakan layers.LSTM(32) setelah Embedding (butuh sedikit lebih lama), bandingkan hasilnya dengan arsitektur GlobalAveragePooling.
c) Ubah threshold prediksi dari 0.5 menjadi 0.6/0.4 — amati perubahan pada precision/recall.
d) Eksperimen preprocessing: coba maxlen lebih panjang (mis. 50) — apakah akurasi meningkat?
e) Analisis kalimat ambiguitas: coba kalimat campur positif-negatif (contoh: "Pengiriman cepat tapi kualitas buruk") dan diskusikan hasilnya.

E. PRAKTIKUM 3 (Prediksi Churn Pelanggan dengan ANN)

1. Tujuan Pembelajaran

- Memahami alur klasifikasi churn pelanggan berbasis Artificial Neural Network (ANN).
- Menyiapkan dataset sintetis yang realistis untuk konteks bisnis digital.
- Melakukan preprocessing (train/test split, standardisasi) dan pelatihan model.
- Mengevaluasi model dengan akurasi, confusion matrix, classification report, dan ROC AUC.
- Mencoba prediksi pelanggan baru dan menginterpretasi hasilnya.

2. Latar Belakang Singkat

Churn adalah kondisi ketika pelanggan berhenti menggunakan layanan (berhenti berlangganan, tidak bertransaksi lagi). Mampu memprediksi churn membantu bisnis melakukan **retensi** (misal: promosi personalisasi, loyalty program) yang lebih tepat sasaran. Pada praktikum ini, kita membangun model **ANN** untuk mengklasifikasikan apakah pelanggan berpotensi **churn (1)** atau **loyal (0)**.

3. Persiapan Lingkungan

Jalankan sel ini untuk melihat versi paket utama yang di butuhkan

```
import sys, numpy, matplotlib, tensorflow, sklearn
print(f'Python       : {sys.version.split()[0]}')
print(f'NumPy        : {numpy.__version__}')
print(f'Matplotlib   : {matplotlib.__version__}')
print(f'Scikit-learn : {sklearn.__version__}')
print(f'TensorFlow   : {tensorflow.__version__}')
```

```
Python       : 3.12.11
NumPy        : 2.0.2
Matplotlib   : 3.10.0
Scikit-learn : 1.6.1
TensorFlow   : 2.19.0
```

4. Membuat Dataset Sintetis Pelanggan

Kita buat dataset sintetis dengan fitur-fitur bisnis yang umum:

- tenure_bulan (lama bergabung),
- avg_monthly_spend (rata-rata belanja/bln),
- trx_per_month (frekuensi transaksi/bln),
- support_tickets (jumlah keluhan/tiket CS),
- promo_user (0/1, apakah sering dapat promo),
- late_payment_rate (proporsi pembayaran telat).

Label churn bernilai **1** jika pelanggan cenderung pergi, **0** jika loyal.

```python
import numpy as np
import pandas as pd

# Random generator untuk reproducibility
rng = np.random.default_rng(2)
n = 800

# Fitur sintetis
tenure_bulan = rng.integers(1, 48, size=n)                # 1 - 48 bulan
avg_monthly_spend = rng.normal(500_000, 150_000, size=n)   # rata-rata pengeluaran
avg_monthly_spend = np.clip(avg_monthly_spend, 50_000, None)  # minimal 50rb
trx_per_month = rng.poisson(8, size=n)                    # ~8 transaksi/bln
support_tickets = rng.poisson(1.2, size=n)                # jumlah keluhan
promo_user = rng.integers(0, 2, size=n)                   # 0/1
late_payment_rate = np.clip(rng.beta(1.5, 6, size=n), 0, 1)  # 0..1

# Probabilitas churn (logistic)
z = (
    -0.04*tenure_bulan
    -0.0000035*avg_monthly_spend
    -0.12*trx_per_month
    +0.35*support_tickets
    +0.9*late_payment_rate
    -0.25*promo_user
    + rng.normal(0, 0.4, size=n)   # noise
)
prob_churn = 1 / (1 + np.exp(-z))

# Gunakan threshold median agar lebih seimbang
threshold = np.median(prob_churn)
churn = (prob_churn > threshold).astype(int)

# Buat DataFrame
df = pd.DataFrame({
    "tenure_bulan": tenure_bulan,
    "avg_monthly_spend": avg_monthly_spend,
    "trx_per_month": trx_per_month,
    "support_tickets": support_tickets,
    "promo_user": promo_user,
    "late_payment_rate": late_payment_rate,
    "churn": churn
})

# Atur tampilan angka desimal 2 digit
pd.options.display.float_format = "{:.2f}".format

# Tampilkan ringkasan awal
print("Dimensi data:", df.shape)
print("\nSebaran label churn (0=loyal, 1=churn):")
print(df['churn'].value_counts())
df.head()
```

```
Dimensi data: (800, 7)

Sebaran label churn (0=loyal, 1=churn):
churn
0    400
1    400
Name: count, dtype: int64
```

	tenure_bulan	avg_monthly_spend	trx_per_month	support_tickets	promo_user	late_payment_rate	churn
0	40	580987.54	5	1	1	0.01	0
1	13	759978.91	7	5	1	0.25	1
2	6	549606.73	7	0	0	0.09	0
3	15	357698.46	9	1	0	0.16	1
4	20	414508.77	7	1	0	0.21	1

5. Ringkasan Data & Pemeriksaan Cepat

Cek dimensi, statistik ringkas, dan sebaran label.

```
# Menampilkan jumlah baris dan kolom dalam dataset
print("Dimensi data:", df.shape)

# Menampilkan ringkasan statistik deskriptif dari fitur numerik
# (mean, std, min, max, quartiles)
print("\nStatistik ringkas (numerik):")
display(df.describe())

# Melihat distribusi label target 'churn'
# value_counts() menghitung berapa banyak data dengan label 0 (loyal) dan 1 (churn)
print("\nSebaran label churn (0=loyal, 1=churn):")
print(df['churn'].value_counts())
```

```
Dimensi data: (800, 7)

Statistik ringkas (numerik):
```

	tenure_bulan	avg_monthly_spend	trx_per_month	support_tickets	promo_user	late_payment_rate	churn
count	800.00	800.00	800.00	800.00	800.00	800.00	800.00
mean	24.23	492029.19	8.01	1.20	0.48	0.20	0.50
std	13.56	151135.46	2.78	1.15	0.50	0.14	0.50
min	1.00	52435.24	0.00	0.00	0.00	0.00	0.00
25%	12.00	395784.19	6.00	0.00	0.00	0.09	0.00
50%	24.00	492573.25	8.00	1.00	0.00	0.17	0.50
75%	36.25	589666.97	10.00	2.00	1.00	0.28	1.00
max	47.00	966523.19	16.00	6.00	1.00	0.84	1.00

```
Sebaran label churn (0=loyal, 1=churn):
churn
0    400
1    400
Name: count, dtype: int64
```

Dataset berisi data dummy pelanggan: umur, frekuensi transaksi, rata-rata nilai transaksi, durasi berlangganan, dsb.

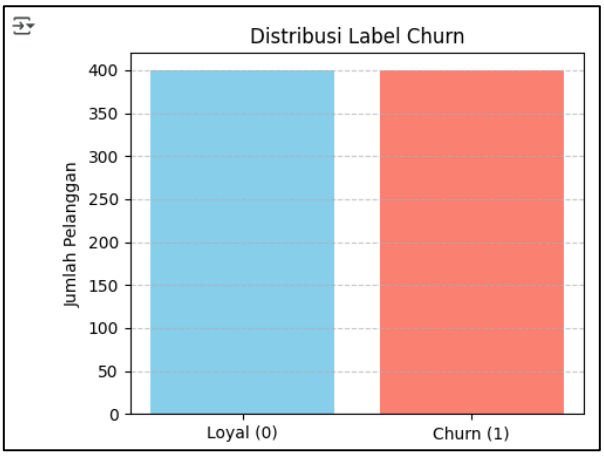
- Label = 1 → pelanggan churn (berhenti).
- Label = 0 → pelanggan loyal (tetap bertahan).

Tujuan: mempelajari pola data pelanggan dan mengajarkan model membedakan pelanggan “churn” vs “loyal”.

6. Visualisasi Sederhana (bar chart churn vs loyal)

Digunakan untuk menggambarkan kondisi sebaran data distribusi label Churn

```
import matplotlib.pyplot as plt

# Hitung jumlah tiap label
label_counts = df['churn'].value_counts()

# Buat bar chart
plt.figure(figsize=(5,4))
plt.bar(label_counts.index, label_counts.values, color=['skyblue','salmon'])
plt.xticks([0,1], ['Loyal (0)','Churn (1)'])
plt.ylabel("Jumlah Pelanggan")
plt.title("Distribusi Label Churn")
plt.grid(axis='y', linestyle='--', alpha=0.7)
plt.show()
```



7. Train/Test Split & Standardisasi Fitur

Kita pisahkan data untuk training dan testing, lalu standarisasi fitur numerik. Langkah ini membersihkan dan menyiapkan data agar siap masuk ke ANN:

- Standarisasi (StandardScaler) → supaya semua fitur punya skala yang seimbang (misalnya umur dalam tahun ≈ 30 vs transaksi ≈ jutaan rupiah). Tanpa ini, fitur besar bisa mendominasi training.
- Split data → data dibagi jadi train dan test (contoh: 80:20). Train dipakai melatih model, test untuk mengevaluasi performa.

Tujuan: data rapi, seimbang, dan representatif untuk proses training.

```
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler

# Pilih fitur yang akan digunakan sebagai input model
features = ["tenure_bulan", "avg_monthly_spend", "trx_per_month",
            "support_tickets", "promo_user", "late_payment_rate"]

# X = variabel independen (fitur), y = target (churn)
X = df[features].values
y = df["churn"].values

# Bagi dataset menjadi train (75%) dan test (25%)
# stratify=y → membagi dengan proporsi label churn yang seimbang antara train & test
X_train, X_test, y_train, y_test = train_test_split(
    X, y, test_size=0.25, random_state=42, stratify=y
)

print("Ukuran data training:", X_train.shape, "Target:", y_train.shape)
print("Ukuran data testing :", X_test.shape, "Target:", y_test.shape)

# Standardisasi fitur numerik agar punya skala rata-rata=0 dan standar deviasi=1
scaler = StandardScaler()

# fit_transform → belajar dari data training lalu terapkan
X_train_scaled = scaler.fit_transform(X_train)

# transform → hanya terapkan standardisasi yang sama ke data test
X_test_scaled  = scaler.transform(X_test)

# Cek ukuran akhir data train dan test
X_train_scaled.shape, X_test_scaled.shape
```

```
Ukuran data training: (600, 6) Target: (600,)
Ukuran data testing : (200, 6) Target: (200,)
((600, 6), (200, 6))
```

Keterangan Output:

- 600 baris data training dengan 6 fitur.
- 200 baris data testing dengan 6 fitur.
- Total = 800 baris (sesuai dataset awal).
- Proses train/test split berhasil (75%:25%).

Semua fitur sudah distandarisasi sehingga setiap kolom punya skala seragam → penting agar ANN tidak bias pada fitur dengan angka besar (misalnya avg_monthly_spend ratusan ribu vs promo_user hanya 0/1).

Kesimpulan:

"Dataset berhasil dibagi menjadi 600 data training dan 200 data testing. Semua fitur sudah dinormalisasi sehingga model ANN akan lebih mudah belajar tanpa dipengaruhi skala angka yang berbeda. Dengan ini, dataset siap digunakan untuk melatih model prediksi churn."

8. Membangun Model ANN (Keras)

Arsitektur sederhana: Dense(32) → Dense(16) → Dense(1) dengan aktivasi ReLU pada hidden layer dan Sigmoid pada output. Tambahkan **EarlyStopping** untuk mencegah overfitting.

a. Membangun Arsitektur ANN

Model ANN dibuat sederhana tapi cukup untuk klasifikasi:

- Input layer → menerima fitur pelanggan (misalnya 4–5 variabel).
- Hidden layer (Dense, ReLU) → menangkap hubungan non-linear antar fitur.
- Output layer (Dense, Sigmoid) → mengeluarkan probabilitas 0–1 (prediksi churn atau loyal).

Tujuan: membangun otak buatan yang bisa "belajar pola" dari data.

b. Kompilasi Model

Model perlu tahu aturan main sebelum dilatih:

- Optimizer: Adam → cara model memperbarui bobot dengan cepat dan efisien.
- Loss: binary_crossentropy → cocok untuk klasifikasi biner (churn vs loyal).
- Metrik: accuracy → mengukur seberapa banyak prediksi yang benar.

Tujuan: menyiapkan model agar bisa belajar dengan parameter yang tepat.

c. Training Model

Model dilatih dengan data train selama beberapa epoch (100).

- Setiap epoch = 1 kali model melihat semua data train.
- Model belajar dengan forward propagation (prediksi), lalu memperbaiki diri lewat backpropagation.
- EarlyStopping digunakan untuk menghentikan training jika tidak ada perbaikan di data validasi → mencegah overfitting.

Tujuan: melatih ANN sampai bisa mengenali pola pelanggan yang churn.

```
import tensorflow as tf
from tensorflow import keras
from tensorflow.keras import layers
from tensorflow.keras.callbacks import EarlyStopping

# Arsitektur ANN:
# Input -> Dense(32, ReLU) -> Dense(16, ReLU) -> Dense(1, Sigmoid)
model = keras.Sequential([
    layers.Input(shape=(X_train_scaled.shape[1],)),  # jumlah fitur (6 kolom)
```

```
    layers.Dense(32, activation='relu'),            # hidden layer 1: 32 neuron
    layers.Dense(16, activation='relu'),            # hidden layer 2: 16 neuron
    layers.Dense(1, activation='sigmoid')           # output biner: probabilitas churn (0..1)
])

# Kompilasi: tentukan "aturan belajar"
model.compile(
    optimizer='adam',               # optimasi cepat & stabil
    loss='binary_crossentropy',     # cocok untuk klasifikasi biner
    metrics=['accuracy']            # pantau akurasi
)

model.summary()  # ringkasan arsitektur + jumlah parameter

# EarlyStopping: hentikan training jika validasi tak membaik
early_stop = EarlyStopping(
    monitor='val_loss',   # pantau loss validasi
    patience=5,           # berhenti jika tak membaik 5 epoch berturut-turut
    restore_best_weights=True  # kembalikan bobot terbaik
)

# Latih model
history = model.fit(
    X_train_scaled, y_train,
    validation_split=0.2,  # 20% dari train jadi data validasi internal
    epochs=100,            # maksimum 100 epoch (bisa berhenti lebih cepat)
    batch_size=32,         # update bobot tiap 32 sampel
    callbacks=[early_stop],
    verbose=0              # diamkan log agar rapi
)

print("Training selesai.")
```

```
Model: "sequential"
```

Layer (type)	Output Shape	Param #
dense (Dense)	(None, 32)	224
dense_1 (Dense)	(None, 16)	528
dense_2 (Dense)	(None, 1)	17

```
Total params: 769 (3.00 KB)
Trainable params: 769 (3.00 KB)
Non-trainable params: 0 (0.00 B)
Training selesai.
```

Interpretasi output:

Tabel menunjukkan setiap layer, bentuk keluarannya, dan jumlah parameter yang dilatih:

1. Dense (32)

- Output Shape: (None, 32) → vektor 32-dim per sampel.
- Param #: 224 → Perhitungan: 6 (fitur input) × 32 (neuron) + 32 (bias) = 224.

2. Dense (16)

- Output Shape: (None, 16)
- Param #: 528 → Perhitungan: 32 × 16 + 16 = 528.

3. Dense (1)

- Output Shape: (None, 1) → probabilitas churn (0..1).
- Param #: 17 → Perhitungan: 16 × 1 + 1 = 17.

Total params: 769 (semuanya trainable). → Artinya, model ini relatif kecil/ringkas—bagus untuk praktik awal, cepat dilatih, dan cukup untuk pola non-linear sederhana.

Makna komponen penting

- ReLU (hidden layers): membantu belajar pola non-linear (hubungan kompleks antar fitur).
- Sigmoid (output): mengubah skor menjadi probabilitas churn.
- EarlyStopping (patience=5): mencegah overfitting; training berhenti saat val_loss tak membaik, dan bobot terbaik dipulihkan.
- validation_split=0.2: menjaga ada “cermin” validasi selama training, terpisah dari test set.

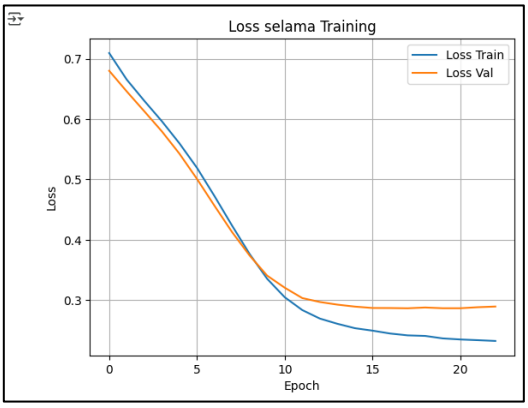
Kesimpulan

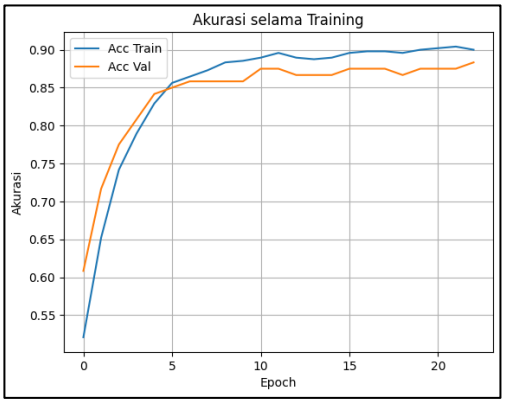
Model sederhana Dense(32) → Dense(16) → Dense(1) dengan ReLU + Sigmoid sudah cukup untuk menunjukkan bagaimana ANN melakukan prediksi churn. Jumlah parameter kecil (769) membuatnya cepat dan pas untuk dataset ini. **EarlyStopping** memastikan hasil yang ditampilkan adalah kinerja terbaik selama pelatihan.

9. Visualisasi Proses Training

Plot ***loss*** dan ***accuracy*** selama training untuk melihat dinamika belajar model.

```
import matplotlib.pyplot as plt

# --- Plot 1: Loss selama training ---
plt.figure()
# loss di data training tiap epoch
plt.plot(history.history['loss'], label='Loss Train')
# loss di data validasi tiap epoch
plt.plot(history.history['val_loss'], label='Loss Val')
plt.title('Loss selama Training')
plt.xlabel('Epoch')    # sumbu x = iterasi epoch
plt.ylabel('Loss')     # sumbu y = nilai loss
plt.legend()
plt.grid(True)
plt.show()

# --- Plot 2: Akurasi selama training ---
plt.figure()
# akurasi di data training
plt.plot(history.history['accuracy'], label='Acc Train')
# akurasi di data validasi
plt.plot(history.history['val_accuracy'], label='Acc Val')
plt.title('Akurasi selama Training')
plt.xlabel('Epoch')
plt.ylabel('Akurasi')
plt.legend()
plt.grid(True)
plt.show()
```





Interpretasi :

a. Grafik Loss (atas)

- Loss Train (biru) turun konsisten dari sekitar 0.7 → 0.22.
- Loss Val (oranye) juga turun hingga stabil di kisaran 0.28.
- Artinya model semakin bisa menyesuaikan bobotnya untuk memprediksi churn dengan baik, dan performa validasi ikut membaik.
- Tidak terlihat tanda overfitting parah, karena loss validasi tidak naik drastis setelah beberapa epoch.

b. Grafik Akurasi (bawah)

- Acc Train (biru) meningkat tajam dari ~0.50 → ~0.90.
- Acc Val (oranye) naik paralel sampai sekitar ~0.87 dan stabil.
- Ini menunjukkan model belajar pola dengan baik dan tidak hanya "menghafal" data training, karena validasi juga cukup tinggi.
- Gap kecil antara train dan val normal → berarti model punya generalization yang cukup baik.

Kesimpulan

- Model ANN churn berhasil belajar dengan baik: akurasi ~90% pada train, ~87% pada validasi.
- Konsistensi loss & akurasi antara train dan validasi → model tidak overfitting.
- EarlyStopping bekerja baik, menghentikan training saat model sudah cukup stabil.

Dengan hasil ini, model bisa dianggap cukup andal untuk prediksi churn pada dataset sintetis.

10. Evaluasi Model pada Data Test

Tampilkan **akurasi**, **confusion matrix**, dan **classification report**.

```
from sklearn.metrics import confusion_matrix, classification_report

# 1) Evaluasi performa di data test (yang tidak pernah dilihat saat training)
#    Menghasilkan nilai loss dan accuracy di test set.
test_loss, test_acc = model.evaluate(X_test_scaled, y_test, verbose=0)
print(f"Akurasi Test: {test_acc*100:.2f}% | Loss Test: {test_loss:.4f}")

# 2) Prediksi probabilitas kelas positif (churn=1) pada data test
y_pred_prob = model.predict(X_test_scaled).ravel()   # ravel -> ubah ke 1D array

# 3) Konversi probabilitas ke label biner dengan threshold 0.5
y_pred = (y_pred_prob >= 0.5).astype(int)

# 4) Confusion Matrix: ringkas benar/salah per kelas
#    Format: [[TN, FP],
#             [FN, TP]]
cm = confusion_matrix(y_test, y_pred)
print("Confusion Matrix:\n", cm)

# 5) Classification Report: precision, recall, f1-score per kelas
#    - precision: dari semua yang DIPREDIKSI positif, berapa yang benar?
#    - recall   : dari semua yang SEBENARNYA positif, berapa yang terdeteksi?
#    - f1-score : rata-rata harmonis precision & recall
print("\nClassification Report:\n", classification_report(y_test, y_pred, digits=4))
```

```
Akurasi Test: 86.50% | Loss Test: 0.3278
7/7 ━━━━━━━━━━━━━━━━━━━━ 0s 10ms/step
Confusion Matrix:
 [[83 17]
 [10 90]]
```

```
Classification Report:
              precision    recall  f1-score   support

           0     0.8925    0.8300    0.8601       100
           1     0.8411    0.9000    0.8696       100

    accuracy                         0.8650       200
   macro avg     0.8668    0.8650    0.8648       200
weighted avg     0.8668    0.8650    0.8648       200
```

Interpretasi Output:
Akurasi Test 86.5% → model cukup andal pada data baru.
a. Confusion Matrix (anggap 0=loyal, 1=churn):

- TN=83: pelanggan loyal diprediksi loyal (benar).
- FP=17: pelanggan loyal diprediksi churn (false alarm).
- FN=10: pelanggan churn diprediksi loyal (kasus terlewat).
- TP=90: pelanggan churn diprediksi churn (benar).

Secara bisnis: hanya 10 pelanggan churn yang lolos (tidak terdeteksi), sedangkan 17 loyal yang "teralaram" churn.

b. Classification Report:
Kelas 0 (loyal)

- Precision 0.8925: dari semua yang diprediksi loyal, 89% benar-benar loyal.
- Recall 0.8300: dari semua loyal, 83% berhasil dikenali (17/100 salah jadi churn).
- F1 0.8601: keseimbangan presisi & cakupan bagus.

Kelas 1 (churn)

- Precision 0.8411: dari semua yang diprediksi churn, 84% benar-benar churn (ada 17 FP).
- Recall 0.9000: dari semua churn, 90% terdeteksi (hanya 10 FN) — ini sangat baik jika tujuan bisnis adalah menangkap calon churn.
- F1 0.8696: performa kuat.

Macro/Weighted Avg ≈ 0.865 → performa seimbang di kedua kelas (karena test set 50:50).

Kesimpulan

- Model generalize dengan baik (test acc 86.5%, F1 ~0.86–0.87 pada kedua kelas).
- Recall churn = 0.90 → sangat bagus untuk use-case mitigasi churn (kita berhasil menemukan 90% pelanggan yang berisiko).
- Jika bisnis ingin mengurangi false positive (menghemat biaya penawaran ke pelanggan loyal), Anda bisa menaikkan threshold (>0.5). Sebaliknya, jika ingin menangkap lebih banyak churn lagi, turunkan threshold sedikit (mis. 0.45) sambil memantau trade-off precision–recall.

11. ROC Curve & AUC

Menggambarkan trade-off **TPR (Recall)** vs **FPR** untuk berbagai ambang (threshold) **Threshold** = batas probabilitas untuk menentukan "positif churn". Contoh: default 0.5 → prediksi ≥0.5 = churn.

Jika threshold **diturunkan** (mis. 0.3):

- Recall (TPR) naik → lebih banyak churn terdeteksi.
- Tapi FPR juga naik → makin banyak pelanggan loyal salah dikira churn.

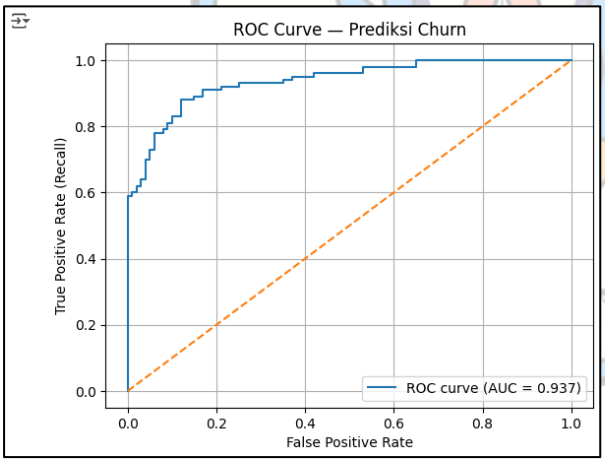
Jika threshold **dinaikkan** (mis. 0.7):

- Recall turun → beberapa churn terlewat.
- Tapi FPR turun → lebih sedikit loyal salah ditandai churn.

Trade-off: kita tidak bisa sekaligus mendapatkan Recall 100% dan FPR 0%. Selalu ada kompromi:

- mau lebih “aman” → Recall tinggi, FPR ikut naik.
- mau lebih “hemat” → FPR rendah, Recall ikut turun.

```python
from sklearn.metrics import roc_curve, auc

# Hitung nilai FPR (False Positive Rate), TPR (True Positive Rate / Recall),
# dan threshold (ambang probabilitas) untuk membuat ROC Curve
fpr, tpr, thresholds = roc_curve(y_test, y_pred_prob)

# Hitung AUC (Area Under Curve) → ringkasan angka kinerja model
roc_auc = auc(fpr, tpr)

# --- Plot ROC Curve ---
plt.figure()
plt.plot(fpr, tpr, label=f'ROC curve (AUC = {roc_auc:.3f})')  # garis biru ROC
plt.plot([0,1], [0,1], linestyle='--')  # garis putus-putus sebagai baseline random
plt.xlabel('False Positive Rate')   # sumbu x: salah prediksi kelas negatif → positif
plt.ylabel('True Positive Rate (Recall)')  # sumbu y: seberapa banyak churn terdeteksi
plt.title('ROC Curve – Prediksi Churn')
plt.legend(loc='lower right')
plt.grid(True)
plt.show()
```



Interpretasi output ROC Curve

1. Garis biru (ROC Curve)

- Menunjukkan kemampuan model membedakan churn vs loyal pada berbagai threshold probabilitas (bukan hanya 0.5).
- Semakin tinggi garis ini mendekati pojok kiri atas → semakin baik model mendeteksi churn dengan minim salah alarm.

2. Garis oranye putus-putus (baseline random guess)

- Ini adalah model acak (tebak-tebakan).
- Jika ROC biru = garis oranye, berarti model tidak lebih baik dari tebak koin.

3. Nilai AUC = 0.937

- Artinya area di bawah kurva ROC adalah 93.7%.
- Semakin mendekati 1.0 → semakin baik.
- Dengan AUC > 0.9, model ini termasuk kategori sangat baik dalam membedakan churn vs loyal.

Kesimpulan:
ROC Curve adalah cara untuk melihat seberapa baik model kita bekerja di berbagai setting threshold, bukan hanya di angka 0.5. Nilai AUC 0.937 menunjukkan model ini sangat mampu membedakan pelanggan churn vs loyal.

12. Prediksi Pelanggan Baru (Contoh)

Masukkan beberapa profil pelanggan untuk memprediksi probabilitas **churn**.

```python
import numpy as np

# Daftar contoh pelanggan baru untuk diuji model
# Format: [tenure, avg_monthly_spend, trx_per_month, support_tickets, promo_user, late_payment_rate]
contoh_pelanggan = [
    [3, 150_000, 2, 3, 0, 0.6],     # Pelanggan1 risiko tinggi churn
    [18, 700_000, 9, 0, 1, 0.1],    # Pelanggan2 risiko rendah churn
    [6, 350_000, 5, 1, 0, 0.2],     # Pelanggan3 menengah
    [30, 1_000_000, 15, 0, 1, 0.05] # Pelanggan4 sangat rendah risiko churn
    ''' tenure: lama berlangganan (bulan)
        spend: rata-rata belanja per bulan (Rp)
        trx: jumlah transaksi per bulan
        tickets: jumlah keluhan ke CS
        promo: apakah dapat promo (0=tidak, 1=ya)
        late_rate: tingkat keterlambatan bayar (0..1)'''
]

# Ubah jadi array numerik (float) agar bisa diproses model
contoh_pelanggan = np.array(contoh_pelanggan, dtype=float)

# Skala fitur (harus sama dengan skala training)
contoh_scaled = scaler.transform(contoh_pelanggan)

# Prediksi probabilitas churn untuk setiap pelanggan
prob = model.predict(contoh_scaled).ravel()

# Tampilkan hasil prediksi
for row, p in zip(contoh_pelanggan, prob):
    label = "Churn" if p >= 0.5 else "Loyal"  # threshold = 0.5
    print(f"{row} -> Prob Churn={p:.3f} => {label}")
```

```
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 37ms/step
[3.0e+00 1.5e+05 2.0e+00 3.0e+00 0.0e+00 6.0e-01] -> Prob Churn=1.000 => Churn
[1.8e+01 7.0e+05 9.0e+00 0.0e+00 1.0e+00 1.0e-01] -> Prob Churn=0.005 => Loyal
[6.0e+00 3.5e+05 5.0e+00 1.0e+00 0.0e+00 2.0e-01] -> Prob Churn=0.996 => Churn
[3.0e+01 1.0e+06 1.5e+01 0.0e+00 1.0e+00 5.0e-02] -> Prob Churn=0.000 => Loyal
```

a. Interpretasi Output:

Pelanggan 1 (risiko tinggi)

- Tenure 3 bulan, spending rendah (150k), sedikit transaksi, banyak keluhan (3 tickets), tanpa promo, sering telat (60%). → Model prediksi 100% churn.
- ✔ Logis: semua indikator negatif.

Pelanggan 2 (risiko rendah)

- Sudah 18 bulan, spending cukup tinggi (700k), transaksi banyak, tidak ada keluhan, sering dapat promo, telat hanya 10%. → Model prediksi 0.5% churn → Loyal.
- ✔ Cocok: indikator sangat sehat.

Pelanggan 3 (risiko menengah)

- Tenure 6 bulan, spending 350k, transaksi moderat, ada 1 keluhan, tidak dapat promo, telat 20%. → Model prediksi 99.6% churn.
- ❗ Menarik: walau datanya "menengah", model tetap melihat risiko tinggi → mungkin dipengaruhi tenure pendek + tanpa promo.

Pelanggan 4 (sangat rendah)

- Tenure 30 bulan, spending tinggi (1 juta), transaksi 15/bln, tidak ada keluhan, dapat promo, telat hanya 5%. → Model prediksi 0% churn → Loyal.
- ✔ Wajar: semua indikator positif.

b. Rekomendasi Tindakan untuk 4 Profil Pelanggan

Pelanggan 1 — Risiko Tinggi (Prob Churn = 1.000)

- Perbaikan layanan cepat: hubungi pelanggan secara proaktif untuk mendengar keluhan dan tawarkan solusi segera.
- Promo/insentif khusus: berikan diskon bulan berikutnya atau voucher gratis ongkir.
- Program onboarding ulang: kirim edukasi “cara memaksimalkan layanan” agar mereka lebih engaged.
- Monitoring intensif: tandai sebagai pelanggan prioritas agar di-follow up oleh tim CS.

Pelanggan 2 — Risiko Rendah (Prob Churn = 0.005)

- Customer loyalty program: tawarkan membership premium atau poin reward.
- Maintain experience: pastikan kualitas layanan tetap konsisten.
- Upsell/cross-sell: karena mereka puas, dorong untuk mencoba produk tambahan (misalnya produk baru/eksklusif).
- Testimonial: minta feedback atau ulasan positif untuk jadi promosi organik.

Pelanggan 3 — Risiko Menengah (Prob Churn = 0.996)

- Targeted promotion: tawarkan kupon khusus atau cashback agar merasa dihargai.
- Atasi keluhan: hubungi kembali terkait masalah yang pernah mereka alami, tunjukkan ada perbaikan.
- Bangun engagement: kirim rekomendasi produk personalisasi sesuai histori transaksi.
- Edukasi manfaat: jelaskan benefit jangka panjang (misalnya hemat biaya, layanan eksklusif).

Pelanggan 4 — Sangat Rendah (Prob Churn = 0.000)

- VIP treatment: jadikan pelanggan ini bagian dari “top-tier loyalty program” (misalnya akses khusus, prioritas CS).
- Retensi jangka panjang: pastikan mereka merasa eksklusif → undangan event, produk baru sebelum publik.
- Referral program: ajak mereka merekomendasikan teman/keluarga, berikan insentif referral.
- Survei kepuasan premium: gunakan mereka sebagai “champion customer” untuk insight produk/layanan baru.

Kesimpulan Bisnis

- Risiko Tinggi (P1, P3): fokus pada retensi aktif (promo, edukasi, perbaikan layanan).
- Risiko Rendah & Sangat Rendah (P2, P4): fokus pada loyalty & upselling, karena mereka sudah puas.

Pendekatan berbasis data ini membuat perusahaan efisien: tidak semua pelanggan diberi promo mahal, hanya yang berisiko churn saja.

13. Latihan & Tantangan

a) Tuning Arsitektur: ubah jumlah neuron (mis. 64–32 atau 16–8) dan tambahkan layers.Dropout(0.2) — amati efeknya pada overfitting.
b) Class Imbalance: jika sebaran label tidak seimbang, coba parameter class_weight saat model.fit(...).
c) Fitur Tambahan: tambahkan fitur avg_session_time atau discount_used_rate ke data sintetis — apakah AUC meningkat?
d) Threshold Bisnis: tentukan threshold khusus untuk kampanye retensi (misal, target Recall ≥ 0.80), lalu evaluasi trade-off precision/recall.
e) Interpretasi Bisnis: tuliskan 5–7 kalimat interpretasi hasil model (fitur mana paling berpengaruh sesuai intuisi domain? strategi retensi apa yang disarankan?).

F. LATIHAN DAN TUGAS

1. Latihan

a) Gambar struktur **Artificial Neural Network (ANN)** sederhana dengan 2 input, 1 hidden layer (4 neuron), dan 1 output. Jelaskan fungsi tiap komponennya dengan bahasa Anda sendiri.

b) Ubah jumlah neuron di hidden layer pada **praktikum XOR** (misalnya dari 4 menjadi 2 atau 8). Amati perbedaan hasil akurasi dan catat perubahan yang terjadi.

c) Pada **praktikum Analisis Sentimen**, coba tambahkan 5 kalimat positif dan 5 kalimat negatif baru. Latih ulang model dan catat perubahan akurasi.

2. Tugas

a) Lanjutkan **praktikum Prediksi Churn Pelanggan** dengan menambahkan minimal 2 fitur baru (misalnya: rata-rata waktu akses aplikasi, jumlah voucher promo yang digunakan). Latih ulang model ANN dan bandingkan hasil evaluasi sebelum dan sesudah penambahan fitur.

b) Bagaimana hasil prediksi churn ini dapat digunakan oleh perusahaan e-commerce untuk merancang strategi **retensi pelanggan**? Berikan contoh aksi nyata (misalnya promo khusus, layanan pelanggan prioritas, atau personalisasi rekomendasi).

c) Buat laporan ringkas (2–3 halaman) berisi:

- Penjelasan dataset dan preprocessing.
- Arsitektur model ANN yang digunakan.
- Hasil evaluasi (akurasi, confusion matrix, ROC AUC).
- Interpretasi bisnis (strategi retensi berdasarkan hasil model).

3. Soal Diskusi

a) Mengapa **Deep Learning** (ANN) lebih baik dalam menangani data teks (NLP) dan data kompleks (gambar, perilaku pelanggan) dibandingkan Machine Learning tradisional?

b) Dalam konteks bisnis digital, mana yang lebih penting: **akurasi tinggi** atau **interpretasi model yang mudah dipahami**? Diskusikan kelebihan dan kekurangannya.

c) Menurut Anda, apa tantangan terbesar dalam menerapkan ANN pada data nyata di perusahaan Indonesia (misalnya keterbatasan data, infrastruktur, atau SDM)?

DAFTAR PUSTAKA

Chatterjee, I. (2021). Machine learning and its application: A quick guide for beginners. Bentham Science Publishers. https://www.researchgate.net/publication/357253428

Fisher, R. (1936). Iris [Dataset]. UCI Machine Learning Repository. https://doi.org/10.24432/C56C76

González, M. L. (2024). Machine learning for libraries with Python libraries: Practical case in the Library of Congress of Chile. South African Journal of Libraries and Information Science, 90(2), 34–45. https://journals.co.za/doi/abs/10.7553/90-2-2406

Jolly, K. (2018). Machine learning with scikit-learn quick start guide: Classification, regression, and clustering techniques in Python. Packt Publishing. https://books.google.com/books?id=28t1DwAAQBAJ

Kharwal, A. (2023). Machine learning algorithms handbook: A step-by-step guide to all machine learning algorithms with implementation using Python!. BPB Publications. https://books.google.com/books?id=gPPUEAAAQBAJ

Laeliyah, A. (2022). Klasifikasi Kepakaran Reviewer Menggunakan Metode ANN (Artificial Neural Network). Repository Universitas Islam Sultan Agung. http://repository.unissula.ac.id/id/eprint/29659

Lavor, V., de Come, F., & dos Santos, M. T. (2024). Machine learning in chemical engineering: Hands-on activities. Education for Chemical Engineers, 47, 32–42. https://www.sciencedirect.com/science/article/pii/S1749772823000477

Lynch, S. (2023). Python for scientific computing and artificial intelligence. CRC Press. https://www.taylorfrancis.com/books/mono/10.1201/9781003285816

Minarno, A. E., Munarko, Y., Akbi, D. R., & Hasanuddin, M. Y. (2022). Workshop Machine Learning Klasifikasi Tumor Otak pada Citra MRI Menggunakan CNN dan SVM. Workshop Data Science, Universitas Muhammadiyah Malang. https://eprints.umm.ac.id/id/eprint/5266/

Raschka, S., & Mirjalili, V. (2019). Python machine learning: Machine learning and deep learning with Python, scikit-learn, and TensorFlow 2. Packt Publishing. https://books.google.com/books?id=sKXIDwAAQBAJ

Rizky, M., Pramuntadi, A., & Prastowo, W. D. (2024). Implementasi Metode Deep Neural Network pada Klasifikasi Penyakit Diabetes Melitus Tipe 2. Journal of Machine Learning and Computer Vision, 3(2), 75–84. https://www.journal.irpi.or.id/index.php/malcom/article/view/1279

Subramanian, V. (2025). Applied machine learning for data science practitioners. Springer. https://books.google.com/books?id=RU5TEQAAQBAJ

Wijaya, W. (2024). Optimasi Model Artificial Neural Network dalam Sistem Klasifikasi Customer Churn Berbasis Web. Repositori Universitas Buddhi Dharma. https://repositori.buddhidharma.ac.id/2591/

Yusup, A., & Aditama, I. (2024). Implementasi Algoritma Regresi Logistik Menggunakan Machine Learning untuk Memprediksi Diabetes. Repositori Universitas Nusa Putra. https://repository.nusaputra.ac.id/id/eprint/1240/