Python untuk Pemula Menuju AI & Machine Learning

Onno W. Purbo
onno@indo.net.id
onno@itts.ac.id
Twitter onnowpurbo

Institut Teknologi Tangerang Selatan
OnnoCenter
2026

Di era AI, belajar Python bukan lagi pilihan tambahan, melainkan langkah penting agar Anda tidak hanya menjadi pengguna teknologi, tetapi juga mampu memahami, membangun, dan mengendalikannya.

Copyright

Karya ini dilisensikan di bawah lisensi **Creative Commons Attribution-ShareAlike 4.0 International (CC BY-SA 4.0).** Artinya, siapapun bebas untuk menggunakan, menyalin, membagikan, dan mengadaptasi materi ini, dengan syarat:

- **Memberikan atribusi yang sesuai** (menyebut sumber asli),
- **Menyertakan lisensi yang sama** jika dimodifikasi atau dikembangkan lebih lanjut.

Lihat detail lisensi di sini https://creativecommons.org/licenses/by-nd/4.0/legalcode.id

Disclaimer: Buku ini disusun untuk tujuan edukasi dan pengantar praktis agar pembaca dapat memahami dasar **Python**, *data analysis*, *machine learning*, dan **AI** secara bertahap. Seluruh materi, contoh kode, dan ilustrasi dalam buku ini dimaksudkan sebagai sarana belajar, sehingga pembaca tetap perlu melakukan verifikasi, pengujian, dan penyesuaian sesuai konteks penggunaan nyata, terutama untuk kebutuhan akademik, profesional, atau produksi.

Abstrak

Buku ini dirancang sebagai **jalan masuk yang terstruktur, praktis, dan relevan** bagi pembaca untuk menguasai fondasi **Python**, **pengolahan data**, **statistik praktis**, **machine learning**, hingga pengantar **AI lanjutan** seperti *deep learning*, *NLP*, dan *LLM*, sehingga pembaca tidak hanya belajar menulis kode, tetapi juga belajar **berpikir dengan data**, memahami alur dari membaca, membersihkan, menganalisis, memvisualisasikan, melatih, dan mengevaluasi model secara bertahap serta sehat secara pedagogis. Tahapan belajar yang ditekankan dalam buku ini bergerak jelas dari **Python dasar → data → statistik → machine learning → horizon AI modern**, agar pembaca tidak terjebak sekadar menyalin kode tanpa memahami logika, kualitas data, *train-test split*, metrik evaluasi, serta perbedaan *supervised* dan *unsupervised learning*. Yang paling penting, buku ini menanamkan bahwa keberhasilan belajar tidak ditentukan oleh menjadi “jenius” sejak awal, melainkan oleh **konsistensi**, **kesabaran**, **kemauan berlatih**, **keberanian mencoba**, dan sikap mental yang sehat untuk membangun fondasi secara kuat sebelum melompat ke topik yang lebih kompleks; dengan sikap seperti itulah pembaca akan lebih siap menjadikan Python sebagai **alat kerja nyata** sekaligus **pintu masuk utama ke dunia AI/ML**.

Kata Pengantar

Alhamdulillah, puji syukur kita panjatkan ke hadirat Allah SWT, karena dengan rahmat dan karunia-Nya buku **Python untuk Pemula Menuju AI & Machine Learning, 2026** ini dapat disusun. Buku ini lahir dari kebutuhan yang semakin nyata: banyak orang ingin masuk ke dunia **AI** dan *machine learning*, tetapi sering merasa terhalang oleh istilah yang rumit, bayangan matematika yang menakutkan, atau kebingungan harus mulai dari mana. Karena itu, buku ini disusun untuk menjadi **pintu masuk yang ramah, bertahap, dan membumi**, agar pembaca dapat memulai perjalanan belajar dengan lebih tenang, terarah, dan percaya diri. Buku ini menempatkan **Python** bukan sekadar sebagai bahasa pemrograman, melainkan sebagai fondasi untuk memahami data, melihat pola, membangun model, membaca hasil, dan melangkah ke dunia AI yang lebih luas.

Terima kasih yang sebesar-besarnya saya sampaikan kepada semua pihak yang telah memberi ruang, inspirasi, dan semangat dalam penyusunan buku ini. Terima kasih kepada rekan-rekan, mahasiswa, praktisi, komunitas *open source*, serta siapa pun yang terus percaya bahwa pembelajaran teknologi harus dibuat **mudah diakses, relevan, dan dapat dipraktikkan**. Semoga buku ini dapat membantu pembaca memahami perjalanan belajar secara sehat: mulai dari **Python dasar**, lalu masuk ke **pengolahan data**, **statistik praktis**, **machine learning**, hingga mengenal arah lanjutan seperti *deep learning*, *NLP*, dan *LLM*. Buku ini juga menekankan bahwa belajar AI tidak perlu tergesa-gesa; yang lebih penting adalah **membangun fondasi yang kuat**, memahami alur kerja dengan benar, dan terus berlatih melalui proyek-proyek kecil yang nyata.

Akhirnya, saya berharap buku ini benar-benar memberi manfaat bagi pembaca pemula, mahasiswa, guru, dosen, praktisi, maupun siapa saja yang ingin bertumbuh di era AI. **Keberhasilan belajar tidak ditentukan oleh harus langsung hebat, tetapi oleh konsistensi, kesabaran, keberanian mencoba, dan kemauan untuk terus memperbaiki diri.** Semoga buku ini menjadi teman belajar yang baik, membuka rasa ingin tahu, memperkuat kepercayaan diri, dan menuntun pembaca untuk melangkah setahap demi setahap menuju pemahaman yang lebih matang. *Semoga bermanfaat.*

Jakarta, Maret 2026
Onno W. Purbo

Daftar Isi

Copyright **3**
Abstrak **4**
Kata Pengantar **5**
Daftar Isi **6**
Bagian I — Mulai dari Nol di Ubuntu 26.04 ..... **36**
Bab 1. Python, AI, dan Jalan Belajar yang Masuk Akal 36
Mengapa Python menjadi bahasa utama untuk data science, AI, dan otomasi 36
Peta perjalanan belajar: Python → data → statistik → ML → AI lanjutan 37
Apa yang realistis dicapai pemula dalam 4–6 bulan 39
Cara memakai buku ini: praktik, eksperimen, mini project 40
Penutup Bab 41
Bab 2. Menyiapkan Ecosystem Python 42
2.1. Dua pilihan ecosystem: Ubuntu 26.04 atau Google Colab 42
1. Ubuntu 26.04: jalur yang lebih kuat dan rapi 42
2. Google Colab: jalur cepat, ringan, dan praktis 43
Rekomendasi praktis untuk pembaca 43
2.2. Jalur 1 — Menyiapkan Python di Ubuntu 26.04 43
1. Instalasi Python di Ubuntu 26.04 44
2. Memahami python3, pip, dan venv 44
3. Membuat proyek pertama dan virtual environment 44
4. Memperbarui pip 45
2.3. VS Code dan Jupyter Notebook di Ubuntu 26.04 45
1. Mengapa Jupyter Notebook penting 45
2. Instal Jupyter Notebook 46
3. Mengapa VS Code penting 46
4. Langkah memakai VS Code 46
2.4. Struktur folder proyek yang rapi 47
2.5. Instal library inti: numpy, pandas, matplotlib, scikit-learn 48
Uji cepat instalasi 48
2.5. Menjalankan file .py 49
Contoh file Python pertama 49
Contoh script kecil dengan pandas 49
2.6. Menjalankan notebook .ipynb 50
Langkah membuat notebook pertama 50
Isi cell pertama 50
Isi cell kedua 50
Isi cell ketiga untuk visualisasi 50
2.7. Jalur 2 — Google Colab untuk pembaca yang ingin cepat mulai 51
Langkah memakai Google Colab 51
Kelebihan Colab untuk pemula 51
Kekurangannya 51
Contoh kode pertama di Colab 51

2.8. Langkah cepat dari nol sampai siap praktik..... 52
Opsi Ubuntu 26.04.....52
Opsi Google Colab..... 52
2.9. Tips menghindari error pemula.....53
Penutup Bab.....53
Bab 3. Python Dasar untuk Calon Praktisi AI.....54
3.1 Mengapa Python dasar penting untuk calon praktisi AI..... 54
3.2 Variabel, tipe data, dan operator..... 55
Apa itu variabel.....55
Mengapa variabel penting..... 55
Contoh di Ubuntu 26.04.....56
Tipe data dasar.....56
Mengecek tipe data..... 57
Operator dasar..... 57
Operator aritmetika.....57
Operator perbandingan..... 58
Operator logika..... 58
Contoh praktis: menghitung nilai akhir..... 58
3.3 if, for, dan while..... 59
if: membuat keputusan..... 59
if, elif, else.....59
Contoh praktis: validasi prediksi sederhana.....59
for: mengulang data.....60
Contoh praktis: menjumlah nilai..... 60
while: mengulang selama kondisi masih benar.....60
Kapan memakai for dan kapan while..... 60
Contoh mini studi kasus..... 61
3.4 list, tuple, set, dan dict.....61
list..... 61
tuple.....62
set.....62
dict.....62
Mengapa dict sangat penting..... 63
Contoh praktis gabungan..... 63
3.5 Fungsi.....63
Apa itu fungsi.....63
Fungsi dengan parameter..... 64
Fungsi dengan return..... 64
Contoh fungsi untuk calon praktisi AI..... 64
Fungsi kecil lebih baik daripada fungsi besar.....64
Contoh fungsi yang lebih realistis.....65
3.6 Membaca file teks, CSV, dan JSON.....65
A. Membaca file teks..... 65
Membaca baris per baris..... 65

Menulis file teks..... 66
B. Membaca CSV..... 66
C. Membaca JSON.....67
Menulis JSON.....67
3.7 Error dasar dan debugging sederhana.....68
A. Jenis error dasar..... 68
1. Syntax error..... 68
2. Runtime error.....68
3. Logical error.....68
B. Memakai try-except.....68
Menangani beberapa jenis error.....69
else dan finally.....69
C. Debugging dengan print()..... 69
D. Pola debugging sederhana yang baik.....70
E. Contoh mini debugging..... 70
3.8 Langkah praktik yang disarankan untuk pembaca..... 70
Tahap 1 — Variabel dan operator.....70
Tahap 2 — if, for, while..... 71
Tahap 3 — Struktur data.....71
Tahap 4 — Fungsi..... 71
Tahap 5 — File handling.....71
Tahap 6 — Debugging.....71
3.9 Mini proyek kecil: data nilai mahasiswa..... 71
Penutup Bab.....73
Bab 4. Mini Project 1 — Python Dasar untuk Mengolah Data Sederhana.....74
Tujuan mini project..... 74
4.1 Skenario kasus: toko kecil dengan data penjualan sederhana..... 75
4.2 Konsep logika sebelum coding.....76
4.3 Struktur data yang dipakai.....76
4.4 Source code Python lengkap..... 77
4.5 Cara menjalankan di Ubuntu 26.04.....80
Tahap 1 — Siapkan folder proyek..... 80
Tahap 2 — Simpan file program.....80
Tahap 3 — Jalankan program..... 80
Tahap 4 — Periksa hasil.....80
Tahap 5 — Buka hasil CSV..... 80
4.6 Contoh keluaran yang diharapkan..... 81
4.7 Bedah kode langkah demi langkah..... 81
1. Menyimpan data awal.....81
2. Menghitung omzet.....81
3. Menambahkan hasil ke setiap item..... 82
4. Menghitung total dan rata-rata..... 82
5. Mencari nilai maksimum dan minimum..... 82
6. Menulis hasil ke CSV.....82

4.8 Isi file CSV yang dihasilkan.....82
File 1 — penjualan_detail.csv.....82
File 2 — ringkasan_penjualan.csv.....83
4.9 Versi lanjutan: membaca data penjualan dari CSV.....83
4.10 Langkah praktik yang disarankan untuk pembaca.....85
Tahap 1 — Jalankan kode apa adanya.....85
Tahap 2 — Pahami logikanya.....85
Tahap 3 — Ubah datanya.....85
Tahap 4 — Tambahkan analisis kecil.....85
Tahap 5 — Ubah output.....86
4.11 Tantangan pengembangan untuk pembaca.....86
4.12 Refleksi: dari coding ke berpikir data.....86
Penutup Bab.....87
Bagian II — Data Analysis dan Statistik yang Benar-Benar Dipakai.....88
Bab 5. NumPy untuk Data Numerik.....88
5.1 Mengapa NumPy penting untuk calon praktisi AI.....88
5.2 Menyiapkan NumPy di Ubuntu 26.04.....89
5.3 Array dan operasi numerik.....89
Apa itu array NumPy.....89
Membuat array 1 dimensi.....89
Membuat array 2 dimensi.....90
Membuat array khusus.....91
Operasi numerik dasar.....91
Operasi dengan skalar.....92
Langkah praktik yang disarankan.....92
5.4 Indexing, slicing, dan reshape.....92
A. Indexing array 1D.....93
B. Slicing array 1D.....93
C. Indexing array 2D.....93
D. Slicing array 2D.....93
E. Mengapa slicing penting.....94
F. reshape(): mengubah bentuk array.....94
G. Contoh reshape(-1, 1).....95
Langkah praktik yang disarankan.....95
5.5 Operasi vektor dan matriks dasar.....96
A. Vektor sebagai array 1D.....96
B. Perkalian elemen-per-elemen.....96
C. Dot product.....96
D. Matriks dasar.....97
E. Perkalian elemen matriks.....97
F. Perkalian matriks sesungguhnya.....98
G. Kenapa ini penting untuk AI/ML.....98
Contoh kecil yang relevan.....99
Langkah praktik yang disarankan.....99

5.6 Statistik sederhana dengan NumPy..... 99
A. Mean, median, variance, standard deviation..... 99
B. Makna intuitif..... 100
C. Statistik pada array 2D berdasarkan sumbu..... 100
D. Contoh praktis: analisis penjualan bulanan..... 100
E. ddof dan simpangan baku sampel..... 101
F. Contoh mini analisis lengkap..... 101
Langkah praktik yang disarankan..... 102
5.7 Source code lengkap: latihan terpadu NumPy..... 102
5.8 Kesalahan umum pemula..... 104
1. Mengira array NumPy sama dengan list Python..... 104
2. Salah memahami shape..... 104
3. Bingung antara * dan @..... 104
4. Salah memilih axis..... 104
5.9 Refleksi: dari angka biasa ke cara berpikir numerik..... 105
Penutup Bab..... 105
Bab 6. Pandas untuk Data Nyata..... 106
6.1 Mengapa Pandas penting untuk calon praktisi AI..... 106
6.2 Menyiapkan Pandas di Ubuntu 26.04..... 107
6.3 Series dan DataFrame..... 108
Apa itu Series..... 108
Contoh Series..... 108
Apa itu DataFrame..... 109
Contoh DataFrame..... 109
Perbedaan Series dan DataFrame..... 109
Contoh lengkap: Series dan DataFrame sekaligus..... 110
Langkah praktik yang disarankan..... 110
6.4 Membaca CSV dan Excel..... 110
A. Membaca CSV..... 110
B. Membaca Excel..... 111
C. Membaca semua sheet Excel..... 111
D. Melihat struktur data setelah dibaca..... 111
E. Menulis kembali ke CSV atau Excel..... 112
Langkah praktik yang disarankan..... 112
6.5 Memilih kolom dan baris, filter, dan sort..... 112
A. Memilih kolom..... 113
B. Memilih baris dengan loc..... 113
C. Memilih baris dengan iloc..... 113
D. Filtering berdasarkan kondisi..... 113
E. Filter dengan lebih dari satu kondisi..... 113
F. Mengurutkan data..... 114
G. Contoh lengkap filter dan sort..... 114
Langkah praktik yang disarankan..... 114
6.6 groupby() dan agregasi..... 114

A. Konsep groupby() 115
B. Rata-rata per grup 115
C. Banyak agregasi sekaligus 115
D. Groupby lebih dari satu kolom 116
E. reset_index() agar lebih rapi 116
Langkah praktik yang disarankan 116
6.7 Merge dan join 116
A. Contoh merge() 117
B. how= pada merge 117
C. Contoh join() 118
D. Kapan memakai merge, kapan join 118
E. Contoh yang dekat dengan dunia nyata 118
Langkah praktik yang disarankan 119
6.8 Ringkasan data cepat 119
A. head() dan tail() 119
B. shape dan columns 119
C. dtypes dan info() 120
D. describe() 120
E. value_counts() 120
F. Contoh alur ringkasan cepat 120
6.9 Source code Python lengkap: latihan terpadu Pandas 121
6.10 Langkah kerja yang disarankan untuk pembaca 123
Tahap 1 — Mulai dari DataFrame kecil 123
Tahap 2 — Masuk ke file nyata 123
Tahap 3 — Belajar memilih data 123
Tahap 4 — Ringkas dan agregasi 123
Tahap 5 — Gabungkan beberapa sumber 123
Tahap 6 — Simpan hasil 123
6.11 Kesalahan umum pemula 124
1. Salah mengira Series dan DataFrame sama 124
2. Lupa bahwa nama kolom sensitif 124
3. Membaca kolom angka sebagai teks 124
4. Bingung antara loc dan iloc 124
5. Salah memilih jenis merge 124
6. Terlalu cepat lanjut tanpa ringkasan awal 124
6.12 Refleksi: dari tabel biasa ke alur kerja data 125
Penutup Bab 126
Bab 7. Membersihkan Data Sebelum Analisis 127
7.1 Mengapa data harus dibersihkan sebelum analisis 127
7.2 Dataset contoh yang akan dipakai 128
7.3 Langkah awal: inspeksi dataset 129
7.4 Missing value 130
Apa itu missing value 130
Langkah 1 — Cek jumlah missing value 130

Langkah 2 — Menghapus baris yang kosong..... 130
Langkah 3 — Mengisi missing value numerik..... 130
Langkah 4 — Mengisi missing value kategorikal.....131
Langkah 5 — Alternatif dengan SimpleImputer.....131
Kapan pakai dropna, kapan pakai fillna.....132
7.5 Data duplikat..... 132
Apa itu data duplikat.....132
Langkah 1 — Cek duplikat..... 132
Langkah 2 — Hapus duplikat..... 132
Langkah 3 — Duplikat berdasarkan kolom tertentu..... 133
Praktik baik.....133
7.6 Tipe data salah.....133
Mengapa tipe data penting.....133
Langkah 1 — Cek tipe data.....133
Langkah 2 — Konversi ke numerik.....134
Langkah 3 — Konversi tanggal bila perlu.....134
Praktik baik.....134
7.7 Outlier dasar.....134
Apa itu outlier.....134
A. Deteksi outlier dengan IQR.....135
B. Deteksi outlier dengan Z-score sederhana..... 135
C. Menangani outlier..... 135
Praktik baik.....136
7.8 Encoding kategori.....136
Mengapa perlu encoding.....136
A. One-hot encoding dengan Pandas..... 137
B. One-hot encoding dengan scikit-learn..... 137
C. Label encoding sederhana.....138
Kapan pakai one-hot, kapan label encoding..... 138
7.9 Standarisasi dan normalisasi..... 138
Mengapa perlu scaling..... 138
A. Normalisasi dengan MinMaxScaler.....139
B. Standardisasi dengan StandardScaler.....139
Kapan pakai normalisasi, kapan standardisasi..... 139
7.10 Source code Python lengkap: membersihkan data langkah demi langkah.....140
7.11 Urutan kerja yang disarankan.....142
7.12 Kesalahan umum pemula.....143
7.13 Refleksi: membersihkan data berarti membangun kepercayaan pada analisis..... 143
Penutup Bab.....144
Bab 8. Visualisasi Data yang Membantu Berpikir..... 145
8.1 Mengapa visualisasi data penting..... 145
8.2 Menyiapkan library visualisasi di Ubuntu 26.04..... 146
8.3 Dataset contoh untuk bab ini.....147

8.4 Prinsip visualisasi yang efektif....147
Prinsip dasar yang perlu dipegang....147
8.5 Line plot....148
Kapan memakai line plot....148
Source code lengkap....148
Apa yang perlu dibaca dari grafik ini....149
Interpretasi contoh....149
Kesalahan umum....149
8.6 Bar plot....149
Kapan memakai bar plot....149
Source code lengkap....149
Apa yang perlu dibaca dari grafik ini....150
Interpretasi contoh....150
Kesalahan umum....150
8.7 Histogram....150
Kapan memakai histogram....150
Source code lengkap....151
Tentang bins....151
Apa yang perlu dibaca....151
Interpretasi contoh....151
Variasi dengan KDE....152
8.8 Boxplot....152
Kapan memakai boxplot....152
Source code lengkap....153
Boxplot per kategori....153
Apa yang perlu dibaca....153
Interpretasi contoh....153
8.9 Scatter plot....154
Kapan memakai scatter plot....154
Source code lengkap....154
Apa yang perlu dibaca....154
Interpretasi contoh....154
Catatan penting....155
8.10 Heatmap korelasi....155
Kapan memakai heatmap korelasi....155
Source code lengkap....155
Apa yang perlu dibaca....155
Interpretasi contoh....155
Batasan....156
8.11 Cara membaca grafik dengan benar....156
A. Selalu mulai dari pertanyaan....156
B. Baca judul, sumbu, dan satuan....156
C. Perhatikan skala....156
D. Bedakan distribusi, tren, dan hubungan....156

E. Jangan terlalu cepat menyimpulkan kausalitas....157
F. Perhatikan outlier....157
G. Cari cerita, bukan sekadar angka....157
8.12 Source code Python lengkap: latihan terpadu visualisasi....157
8.13 Langkah kerja yang disarankan untuk pembaca....159
8.14 Kesalahan umum pemula....159
8.15 Refleksi: visualisasi adalah alat berpikir....160
Penutup Bab....160
Bab 9. Statistik Deskriptif untuk Pemula....161
9.1 Mengapa statistik deskriptif penting sebelum modeling....161
9.2 Dataset contoh untuk bab ini....162
9.3 Langkah awal: membaca dan mengenali data....163
9.4 Mean, median, modus....164
A. Mean....164
B. Median....164
C. Modus....164
D. Kapan mean, median, dan modus dipakai....165
E. Source code lengkap....165
9.5 Variansi dan standar deviasi....165
A. Mengapa perlu ukuran sebaran....165
B. Variansi....165
C. Standar deviasi....166
D. Cara membacanya....166
E. Source code lengkap....166
9.6 Persentil, kuartil, dan IQR....166
A. Persentil....166
B. Kuartil....167
C. IQR....167
D. Menggunakan IQR untuk outlier....168
E. Kapan IQR berguna....168
9.7 Distribusi data....168
A. Apa itu distribusi....168
B. Histogram untuk distribusi....169
C. Boxplot untuk distribusi dan outlier....169
D. Skewness....169
E. Cara membaca distribusi....169
9.8 Korelasi Pearson dan Spearman....170
A. Mengapa korelasi penting....170
B. Pearson correlation....170
C. Spearman correlation....170
D. Kapan pakai Pearson, kapan Spearman....171
E. Korelasi matriks....171
F. Cara membaca korelasi....171
9.9 Source code Python lengkap: statistik deskriptif terpadu....172

9.10 Kapan statistik membantu sebelum modeling....174
Contoh konkret....174
Singkatnya....174
9.11 Urutan kerja yang disarankan....175
9.12 Kesalahan umum pemula....175
9.13 Refleksi: statistik deskriptif adalah bahasa pertama data....175
Penutup Bab....176
Bab 10. Regresi Linear sebagai Jembatan ke Machine Learning....177
10.1 Mengapa regresi linear penting....177
10.2 Intuisi hubungan antar variabel....178
Intuisi sederhana....179
Hubungan tidak selalu sempurna....179
10.3 Dataset contoh untuk bab ini....179
Dataset 1 — Regresi linear sederhana....179
Dataset 2 — Regresi linear berganda....180
10.4 Regresi linear sederhana....180
Konsep....180
Langkah kerja yang benar....181
Source code lengkap....181
10.5 Koefisien, intercept, dan residual....182
A. Koefisien....182
B. Intercept....183
C. Residual....183
Visualisasi residual sederhana....183
10.6 Visualisasi garis regresi....184
Mengapa visualisasi ini penting....184
Versi visual yang lebih rapi....184
10.7 Regresi linear berganda....185
Konsep....185
Intuisi....186
Source code lengkap....186
10.8 Evaluasi dasar: MAE, MSE, RMSE, dan R^2....187
A. MAE — Mean Absolute Error....187
B. MSE — Mean Squared Error....187
C. RMSE — Root Mean Squared Error....187
D. R^2 — Coefficient of Determination....188
10.9 Train-test split agar lebih realistis....188
Source code lengkap....188
10.10 Contoh interpretasi hasil model....190
10.11 Asumsi praktis dan keterbatasan regresi linear....190
Regresi linear cocok jika:....190
Regresi linear kurang cocok jika:....190
Tanda model linear mungkin kurang cocok....191
10.12 Source code Python lengkap: latihan terpadu Bab 10....191

10.13 Langkah kerja yang disarankan untuk pembaca....193
10.14 Kesalahan umum pemula....194
10.15 Refleksi: mengapa regresi linear adalah jembatan yang sangat baik....194
Penutup Bab....195
Bab 11. Statistik Inferensial Ringan untuk Data Science....196
11.1 Mengapa statistik inferensial penting untuk data science....196
11.2 Sampel vs populasi....197
A. Apa itu populasi....197
B. Apa itu sampel....197
C. Mengapa kita memakai sampel....197
D. Resiko memakai sampel....198
E. Ilustrasi intuitif....198
F. Contoh source code Python lengkap....198
G. Pelajaran dari contoh ini....199
11.3 Konsep probabilitas dasar....199
A. Probabilitas sebagai ukuran ketidakpastian....199
B. Probabilitas empiris....199
C. Event, komplemen, dan probabilitas bersyarat....199
D. Kenapa probabilitas penting untuk data science....200
E. Source code Python lengkap: simulasi probabilitas sederhana....200
F. Intuisi yang harus ditangkap....200
11.4 Distribusi sampel, law of large numbers, dan central limit theorem....200
A. Law of Large Numbers....200
B. Distribusi sampel....201
C. Central Limit Theorem secara intuitif....201
D. Mengapa ini penting....201
E. Source code Python lengkap: simulasi distribusi rata-rata sampel....201
F. Pelajaran dari kode ini....202
11.5 Uji sederhana secara intuitif....202
A. Apa itu uji hipotesis....202
B. Intuisi H0 dan H1....202
C. Apa itu p-value secara sederhana....203
D. Penting: “menolak H0” bukan berarti “membuktikan H1 mutlak”....203
E. Contoh uji satu sampel secara intuitif....203
F. Kapan uji sederhana ini berguna....204
11.6 Contoh uji perbedaan dua kelompok secara intuitif....204
Source code Python lengkap....204
Intuisi yang harus dipahami....205
11.7 Mengapa statistik penting untuk interpretasi, bukan sekadar hitung....205
A. Statistik membantu interpretasi....205
B. Statistik membantu membatasi ego analitis....205
C. Statistik dan machine learning saling melengkapi....206
D. Contoh konkret....206
E. Inti pesan bab ini....206

11.8 Source code Python lengkap: bab terpadu..... 206
11.9 Langkah kerja yang disarankan untuk pembaca..... 208
11.10 Kesalahan umum pemula..... 208
11.11 Refleksi: statistik inferensial melatih kerendahan hati analitis..... 209
Penutup Bab..... 209
Bab 12. Mini Project 2 — EDA dan Statistik Lengkap..... 210
12.1 Tujuan mini project..... 210
12.2 Studi kasus mini project..... 211
Skenario..... 211
12.3 Menyiapkan dataset..... 211
12.4 Langkah 1 — Load dataset..... 212
Apa yang harus diperhatikan..... 213
12.5 Langkah 2 — Cleaning..... 213
A. Tujuan cleaning pada mini project ini..... 213
B. Source code lengkap untuk cleaning..... 213
Apa yang dilatih di tahap ini..... 215
12.6 Langkah 3 — Statistik deskriptif..... 215
Source code lengkap..... 215
Cara membaca hasilnya..... 216
Tujuan tahap ini..... 216
12.7 Langkah 4 — Korelasi..... 216
Pertanyaan analitis..... 216
Source code lengkap..... 216
Cara membaca korelasi..... 217
Catatan penting..... 217
12.8 Langkah 5 — Regresi linear sederhana..... 217
Pertanyaan model..... 217
Source code lengkap..... 217
Apa yang dipelajari di tahap ini..... 219
12.9 Langkah 6 — Insight awal..... 219
Contoh insight awal yang mungkin muncul..... 219
Hal yang perlu dijaga..... 220
12.10 Langkah 7 — Menulis narasi temuan..... 220
Struktur narasi temuan yang baik..... 220
Contoh narasi temuan..... 221
Kenapa narasi penting..... 221
12.11 Source code Python lengkap: mini project terpadu..... 221
12.12 Alur kerja yang sebaiknya dibiasakan..... 224
12.13 Kesalahan umum pemula..... 224
12.14 Refleksi: dari kode ke cara berpikir analitis..... 225
Penutup Bab..... 225
Bagian III — Synthetic Data, Feature, dan Persiapan Modeling..... 226
Bab 13. Men-generate Synthetic Data dengan Python..... 226
13.1 Apa itu synthetic data dan kapan dipakai..... 226

Kapan synthetic data dipakai....227
Mengapa penting untuk pemula....227
13.2 Menyiapkan lingkungan kerja....227
13.3 Membuat data numerik buatan dengan NumPy....228
A. Membuat data numerik sederhana....228
B. Menambahkan variasi kecil....228
C. Visualisasi hasil....229
Apa yang dipelajari dari langkah ini....229
13.4 Membuat dataset regresi dengan make_regression....229
Apa yang dihasilkan....229
Source code lengkap....230
Menambah fitur lebih banyak....231
Apa yang bisa dipelajari....231
13.5 Membuat dataset klasifikasi dengan make_classification....231
Cocok untuk latihan seperti:....231
Source code lengkap....231
Versi tiga kelas....232
Apa yang dipelajari....233
13.6 Membuat data clustering dengan make_blobs....233
Kapan dipakai....233
Source code lengkap....233
Mengubah tingkat sebaran cluster....234
Apa yang dipelajari....234
13.7 Menambah noise untuk simulasi kasus nyata....234
Mengapa noise penting....234
A. Menambah noise pada data regresi NumPy....235
B. Membandingkan noise kecil dan besar....235
Pelajaran penting....236
13.8 Source code Python lengkap: latihan terpadu synthetic data....236
13.9 Kelebihan synthetic data....238
Kelebihan utama....238
Kelebihan untuk pemula....239
13.10 Keterbatasan synthetic data....239
Keterbatasan utama....239
Pelajaran penting....239
13.11 Kapan synthetic data sangat membantu....239
Penutup Bab....240
Bab 14. Feature Engineering Dasar....241
14.1 Mengapa feature engineering penting....241
Contoh intuitif....242
14.2 Dataset contoh untuk bab ini....242
14.3 Memilih feature yang relevan....243
Apa itu feature selection secara dasar....243
Prinsip memilih fitur....243

Contoh memilih fitur dari dataset mahasiswa....243
Source code lengkap....243
Latihan berpikir....244
14.4 Membuat feature turunan sederhana....244
Contoh fitur turunan pada dataset mahasiswa....245
Source code lengkap....245
Kapan fitur turunan berguna....245
Catatan penting....245
14.5 Encoding kategori....246
A. One-hot encoding untuk fitur kategori....246
B. Label encoding sederhana untuk target biner....246
Kapan pakai yang mana....247
Catatan praktis....247
14.6 Scaling....247
Mengapa scaling penting....247
Model yang sensitif terhadap skala....247
A. StandardScaler....247
B. MinMaxScaler....248
Kapan pakai standardisasi, kapan normalisasi....248
14.7 Split train/test....248
Mengapa perlu train-test split....248
Source code lengkap....249
Catatan penting....249
14.8 Menghindari data leakage....249
Apa itu data leakage....249
Contoh kebocoran yang sering terjadi....250
Contoh praktik salah....250
Contoh praktik yang benar....250
Inti kebiasaan sehat....251
14.9 Pipeline preprocessing sederhana....251
Source code lengkap....251
Mengapa ini bagus untuk pemula....252
14.10 Source code Python lengkap: latihan terpadu Bab 14....252
14.11 Urutan kerja yang disarankan....254
14.12 Kesalahan umum pemula....254
14.13 Refleksi: feature engineering adalah seni dan disiplin....255
Penutup Bab....255
Bagian IV — Machine Learning Inti dengan Banyak Contoh Kode....256
Bab 15. Memahami Machine Learning tanpa Takut Istilah....256
15.1 AI, ML, dan deep learning....256
Ilustrasi intuitif....257
Contoh sehari-hari....257
Source code Python kecil: AI/ML bukan sihir....257
15.2 Supervised vs unsupervised learning....258

A. Supervised learning..... 258
B. Unsupervised learning..... 258
Ringkasnya.....258
Source code Python lengkap: contoh supervised sederhana..... 259
Source code Python lengkap: contoh unsupervised sederhana..... 259
15.3 Klasifikasi vs regresi.....259
A. Regresi.....260
B. Klasifikasi..... 260
Ringkasan sederhana.....260
Source code Python lengkap: regresi sederhana.....260
Source code Python lengkap: klasifikasi sederhana..... 261
15.4 Overfitting vs underfitting.....261
A. Overfitting.....261
B. Underfitting.....262
Ilustrasi intuitif.....262
Source code Python lengkap: ilustrasi sederhana..... 262
Pelajaran dari contoh ini.....263
15.5 Train, validation, test..... 263
A. Training set.....263
B. Validation set.....263
C. Test set.....263
Rasio umum..... 263
Source code Python lengkap: train-test split..... 264
Source code Python lengkap: train-validation-test sederhana..... 264
15.6 Kenapa model harus dievaluasi..... 265
Mengapa evaluasi wajib..... 265
Metrik tergantung tugas.....265
Source code Python lengkap: evaluasi regresi..... 266
Source code Python lengkap: evaluasi klasifikasi.....267
15.7 Contoh alur ML sederhana dari awal sampai evaluasi.....267
15.8 Cara berpikir yang sehat untuk pemula.....269
15.9 Kesalahan umum pemula.....269
15.10 Refleksi: ML bukan soal istilah, tetapi soal alur pikir..... 269
Penutup Bab.....270
Bab 16. Workflow Machine Learning End-to-End..... 271
16.1 Mengapa workflow lebih penting daripada sekadar algoritma..... 271
16.2 Gambaran besar alur machine learning end-to-end.....272
16.3 Dataset contoh..... 272
16.4 Definisikan masalah..... 273
Pertanyaan yang harus dijawab..... 273
Contoh pada dataset mahasiswa..... 273
16.5 Tentukan target.....273
Pada contoh ini.....274
Kesalahan umum.....274

16.6 Pilih feature..... 274
Pada contoh ini.....274
Prinsip memilih fitur..... 275
16.7 Split data..... 275
Kenapa split penting.....275
Source code Python lengkap.....275
16.8 Bangun baseline.....276
Kenapa baseline penting.....276
Bentuk baseline yang umum..... 276
Source code Python lengkap: baseline mean..... 276
16.9 Latih model.....277
Source code Python lengkap.....277
Apa yang dipelajari..... 277
16.10 Evaluasi.....277
Source code Python lengkap.....278
Apa yang harus dibaca.....279
16.11 Iterasi.....279
Apa arti iterasi.....279
Contoh iterasi sederhana..... 279
Source code Python lengkap: baseline vs model.....279
Makna iterasi..... 280
16.12 Simpan model sederhana..... 280
Kenapa model disimpan.....280
Source code Python lengkap.....281
Memuat kembali model..... 281
16.13 Source code Python lengkap: workflow end-to-end..... 282
16.14 Dokumentasi eksperimen.....284
Hal yang sebaiknya dicatat.....284
16.15 Kesalahan umum pemula.....284
16.16 Refleksi: ML yang baik adalah proses, bukan trik..... 285
Penutup Bab.....285
Bab 17. Bagaimana Memilih Algoritma yang Cocok?..... 286
17.1 Mulai dari pertanyaan, bukan dari nama algoritma..... 286
17.2 Jika target berupa angka → regresi..... 287
Contoh masalah regresi..... 287
Algoritma awal yang masuk akal.....287
Kapan mulai dari Linear Regression..... 287
Source code Python lengkap: regresi sederhana.....287
17.3 Jika target berupa kelas → klasifikasi..... 288
Contoh masalah klasifikasi.....288
Algoritma awal yang masuk akal.....288
Kapan mulai dari Logistic Regression..... 289
Source code Python lengkap: klasifikasi sederhana..... 289
17.4 Jika tidak ada label → clustering.....290

Contoh kasus clustering..... 290
Kapan K-Means cocok..... 290
Catatan penting tentang K-Means.....290
Source code Python lengkap: clustering dengan K-Means.....291
17.5 Jika butuh model mudah dijelaskan..... 291
Kenapa model-model ini mudah dijelaskan.....291
Source code Python lengkap: Decision Tree yang mudah dijelaskan.....292
17.6 Jika ingin baseline cepat..... 292
KNN cocok jika:..... 292
Decision Tree cocok jika:.....292
Source code Python lengkap: baseline KNN.....293
17.7 Jika data tabular cukup kompleks..... 293
Kapan Random Forest cocok.....294
Kapan Gradient Boosting cocok.....294
Catatan praktis..... 294
Source code Python lengkap: Random Forest sederhana.....294
17.8 Jika ingin segmentasi → K-Means.....295
Cocok untuk:.....295
Tetapi ingat..... 295
17.9 Perbandingan sederhana: interpretabilitas, akurasi, kompleksitas, kebutuhan preprocessing.....295
1. Linear Regression / Logistic Regression.....295
2. Decision Tree.....296
3. KNN.....296
4. Random Forest.....296
5. Gradient Boosting.....297
6. K-Means..... 297
17.10 Aturan cepat memilih algoritma.....298
17.11 Source code Python lengkap: memilih algoritma dari jenis masalah..... 299
17.12 Refleksi: memilih algoritma adalah soal kompromi, bukan dogma..... 300
Penutup Bab.....300
Bab 18. Contoh Pemilihan Algoritma + Source Code..... 301
18.1 Cara membaca bab ini..... 301
18.2 Kasus 1 — Prediksi harga rumah → LinearRegression.....302
Mengapa LinearRegression cocok.....302
Kapan LinearRegression menjadi pilihan bagus..... 302
Dataset contoh..... 302
Source code Python lengkap.....303
Apa yang dipelajari dari kasus ini.....304
18.3 Kasus 2 — Lulus/tidak lulus → LogisticRegression..... 304
Mengapa LogisticRegression cocok.....304
Kapan LogisticRegression menjadi pilihan bagus.....304
Dataset contoh..... 304
Source code Python lengkap.....305

Apa yang dipelajari dari kasus ini....306
18.4 Kasus 3 — Klasifikasi pelanggan → RandomForestClassifier....306
Mengapa RandomForestClassifier cocok....306
Kapan RandomForestClassifier menjadi pilihan bagus....306
Dataset contoh....306
Source code Python lengkap....307
Apa yang dipelajari dari kasus ini....308
18.5 Kasus 4 — Segmentasi pelanggan → KMeans....308
Mengapa KMeans cocok....308
Kapan KMeans menjadi pilihan bagus....308
Dataset contoh....308
Source code Python lengkap....309
Apa yang dipelajari dari kasus ini....309
18.6 Template kode umum: pilih model, fit, prediksi, evaluasi....310
Template umum untuk supervised learning....310
Template umum untuk clustering....311
Cara memakai template ini....311
18.7 Cara memilih model dari empat contoh ini....311
18.8 Kesalahan umum pemula saat memilih algoritma....312
18.9 Refleksi: memilih algoritma adalah bagian dari berpikir analitis....312
Penutup Bab....313
Bab 19. Supervised Learning I — Regresi....314
19.1 Apa itu regresi dalam supervised learning....314
Ciri masalah regresi....314
Contoh masalah regresi....315
Alur umum regresi....315
19.2 Dataset contoh untuk bab ini....315
19.3 Linear Regression....316
Intuisi....316
Kapan memakai Linear Regression....316
Source code Python lengkap....317
Apa yang dipelajari....318
19.4 Ridge Regression....318
Intuisi....318
Kapan memakai Ridge Regression....318
Catatan penting....318
Source code Python lengkap....319
Apa yang dipelajari....319
19.5 Lasso Regression....320
Intuisi....320
Kapan memakai Lasso Regression....320
Source code Python lengkap....321
Apa yang dipelajari....322
19.6 Polynomial Regression secara ringan....322

Mengapa perlu Polynomial Regression....322
Kapan memakai Polynomial Regression....322
Catatan penting....322
Dataset contoh nonlinier ringan....322
Source code Python lengkap....323
Apa yang dipelajari....324
19.7 Kapan memakai masing-masing....324
1. Linear Regression....324
2. Ridge Regression....324
3. Lasso Regression....325
4. Polynomial Regression....325
19.8 Perbandingan intuitif....325
19.9 Source code Python lengkap: perbandingan empat model....326
19.10 Kesalahan umum pemula....327
19.11 Refleksi: regresi adalah laboratorium yang sangat baik untuk belajar model 328
Penutup Bab....328
Bab 20. Evaluasi Model Regresi....329
20.1 Mengapa evaluasi model regresi itu penting....329
20.2 Dataset contoh untuk bab ini....330
20.3 Menyiapkan model awal untuk dievaluasi....331
20.4 MAE....332
Apa itu MAE....332
Kelebihan MAE....332
Source code Python lengkap....332
Cara membaca MAE....333
20.5 MSE....333
Apa itu MSE....333
Karakter MSE....333
Source code Python lengkap....334
Cara membaca MSE....334
20.6 RMSE....334
Apa itu RMSE....334
Kelebihan RMSE....335
Source code Python lengkap....335
Cara membaca RMSE....335
20.7 R^2....336
Apa itu R^2....336
Intuisi sederhana....336
Catatan penting....336
Source code Python lengkap....336
Cara membaca R^2....337
20.8 Menghitung semua metrik sekaligus....337
20.9 Visualisasi prediksi vs aktual....338
A. Scatter prediksi vs aktual....338

Cara membaca grafik ini....338
B. Versi dengan garis diagonal pembanding....339
20.10 Error analysis dasar....339
A. Menghitung residual....339
Source code Python lengkap....340
B. Plot residual....341
Cara membaca residual....341
20.11 Cross-validation sederhana....341
A. Intuisi k-fold cross-validation....342
B. Source code Python lengkap....342
C. Kenapa neg_mean_squared_error negatif....342
D. Kapan cross-validation berguna....343
20.12 Source code Python lengkap: evaluasi model regresi terpadu....343
20.13 Cara membaca hasil evaluasi dengan sehat....345
20.14 Kesalahan umum pemula....345
20.15 Refleksi: evaluasi model adalah latihan kejujuran analitis....346
Penutup Bab....346
Bab 21. Supervised Learning II — Klasifikasi Dasar....347
21.1 Apa itu klasifikasi....347
Intuisi sederhana....347
21.2 Dataset contoh untuk bab ini....348
21.3 Metrik evaluasi klasifikasi yang akan dipakai....348
Metrik dasar....349
Fungsi evaluasi umum....349
21.4 Logistic Regression....349
Intuisi....349
Kapan memakai Logistic Regression....349
Source code Python lengkap....350
Apa yang dipelajari....351
21.5 K-Nearest Neighbors (KNN)....351
Intuisi....351
Kapan memakai KNN....351
Kenapa scaling penting....352
Source code Python lengkap....352
Apa yang dipelajari....353
21.6 Decision Tree....353
Intuisi....353
Kapan memakai Decision Tree....353
Source code Python lengkap....354
Apa yang dipelajari....355
21.7 Random Forest....355
Intuisi....355
Kapan memakai Random Forest....355
Source code Python lengkap....356

Apa yang dipelajari....357
21.8 Perbandingan kelebihan dan kekurangan....357
Logistic Regression....357
KNN....357
Decision Tree....357
Random Forest....358
21.9 Kapan memakai masing-masing....358
Mulai dari Logistic Regression jika:....358
Coba KNN jika:....358
Coba Decision Tree jika:....358
Coba Random Forest jika:....359
21.10 Source code Python lengkap: perbandingan empat model....359
21.11 Alur kerja yang disarankan untuk pembaca....360
21.12 Kesalahan umum pemula....360
21.13 Refleksi: klasifikasi dasar adalah laboratorium yang sangat baik....361
Penutup Bab....361
Bab 22. Evaluasi Model Klasifikasi....362
22.1 Mengapa evaluasi model klasifikasi tidak boleh hanya memakai accuracy....362
22.2 Dataset contoh untuk bab ini....363
22.3 Menyiapkan model awal untuk dievaluasi....364
22.4 Confusion matrix....365
Apa itu confusion matrix....365
Source code Python lengkap....365
Cara membaca confusion matrix....366
22.5 Accuracy....366
Apa itu accuracy....366
Kapan accuracy berguna....366
Source code Python lengkap....367
Cara membaca accuracy....367
22.6 Precision....367
Apa itu precision....367
Kapan precision penting....368
Source code Python lengkap....368
Cara membaca precision....369
22.7 Recall....369
Apa itu recall....369
Kapan recall penting....369
Source code Python lengkap....370
Cara membaca recall....370
22.8 F1-score....370
Apa itu F1-score....370
Kapan F1-score penting....371
Source code Python lengkap....371
Cara membaca F1-score....371

22.9 classification_report() sebagai ringkasan cepat..... 372
Source code Python lengkap.....372
Apa itu support..... 372
22.10 ROC-AUC secara intuitif..... 373
A. Apa itu ROC.....373
B. Apa itu AUC..... 373
C. Intuisi sederhana.....373
D. Source code Python lengkap.....374
E. Cara membaca ROC-AUC secara intuitif.....374
F. Catatan penting tentang ROC-AUC.....374
22.11 Class imbalance dasar..... 375
Apa itu class imbalance.....375
Kenapa ini berbahaya.....375
Contoh dataset imbalanced sederhana.....375
Source code Python lengkap.....376
Pelajaran dari contoh ini.....376
22.12 Source code Python lengkap: evaluasi klasifikasi terpadu.....377
22.13 Cara membaca hasil evaluasi dengan sehat..... 379
22.14 Kesalahan umum pemula.....379
22.15 Refleksi: evaluasi klasifikasi adalah soal memahami konsekuensi kesalahan.... 380
Penutup Bab.....380
Bab 23. Unsupervised Learning I — Clustering..... 381
23.1 Apa itu clustering.....381
Intuisi sederhana..... 382
Kapan clustering dipakai..... 382
23.2 K-Means..... 382
Apa itu K-Means.....382
Kenapa K-Means populer.....383
Catatan penting tentang K-Means.....383
23.3 Dataset contoh untuk bab ini.....383
23.4 Langkah awal: memuat dan melihat data.....384
23.5 Mengapa scaling penting sebelum K-Means..... 384
Source code Python lengkap.....385
Intuisi..... 385
23.6 Menjalankan K-Means.....385
Apa yang terjadi di kode ini..... 386
23.7 Memilih jumlah cluster.....386
A. Elbow Method..... 386
Cara membaca Elbow Method..... 387
B. Silhouette Score.....387
Cara memakai dua pendekatan ini bersama.....388
23.8 Visualisasi cluster.....388
Source code Python lengkap.....389

Apa yang harus diperhatikan....389
23.9 Studi kasus segmentasi pelanggan....390
Masalah....390
Alur kerja studi kasus....390
Source code Python lengkap: studi kasus terpadu....391
23.10 Cara membaca hasil segmentasi....392
23.11 Kelebihan dan keterbatasan K-Means....392
Kelebihan K-Means....392
Keterbatasan K-Means....393
Kapan perlu mempertimbangkan algoritma lain....393
23.12 Kesalahan umum pemula....393
23.13 Refleksi: clustering adalah alat eksplorasi, bukan mesin kebenaran....394
Penutup Bab....394
Bab 24. Unsupervised Learning II — Reduksi Dimensi....395
24.1 Mengapa dimensi perlu dikurangi....395
24.2 Apa itu PCA secara intuitif....396
Intuisi sederhana....396
Apa yang dipertahankan PCA....396
24.3 PCA bukan sekadar memilih kolom....397
24.4 Dataset contoh untuk bab ini....397
24.5 Langkah awal: memuat dan melihat data....398
24.6 Scaling sebelum PCA....398
Mengapa scaling penting....398
Source code Python lengkap....399
24.7 Menjalankan PCA ke 2 dimensi....399
Source code Python lengkap....399
Apa itu explained_variance_ratio_....400
24.8 Visualisasi 2D dari data berdimensi lebih tinggi....400
Source code Python lengkap....400
Apa yang bisa dibaca dari visualisasi ini....401
24.9 Studi kecil: PCA sebelum clustering....401
Contoh konsep....401
Source code Python lengkap....402
Apa yang dipelajari dari contoh ini....402
24.10 Kapan PCA sangat berguna....403
24.11 Keterbatasan PCA....403
24.12 Source code Python lengkap....404
24.13 Cara membaca hasil PCA dengan sehat....405
Perhatikan explained_variance_ratio_....405
Jangan terlalu cepat menganggap PC1 dan PC2 punya arti yang jelas....405
Gunakan PCA terutama untuk:....405
Jangan lupa konteks....406
24.14 Kesalahan umum pemula....406
24.15 Refleksi: reduksi dimensi adalah seni merangkum tanpa terlalu banyak

kehilangan..... 406
Penutup Bab..... 407
Bab 25. Mini Project 3 — Regresi End-to-End..... 408
25.1 Tujuan mini project..... 408
25.2 Studi kasus: prediksi harga rumah..... 409
Skenario..... 409
Kenapa kasus ini cocok..... 409
25.3 Menyiapkan dataset..... 409
Masalah yang sengaja disisipkan..... 410
25.4 Langkah 1 — Load dataset..... 411
Apa yang harus dilihat..... 411
25.5 Langkah 2 — Cleaning..... 411
Source code Python lengkap..... 412
Catatan penting..... 413
25.6 Langkah 3 — Feature engineering dasar..... 413
Prinsip memilih feature..... 413
Kita juga perlu:..... 414
Contoh fitur turunan..... 414
Source code Python lengkap..... 414
Kenapa langkah ini penting..... 415
25.7 Langkah 4 — Train model..... 415
Source code Python lengkap..... 415
Apa yang dipelajari di tahap ini..... 416
25.8 Langkah 5 — Evaluasi..... 416
Source code Pytho n lengkap..... 417
Apa yang harus dibaca..... 418
25.9 Langkah 6 — Interpretasi hasil..... 418
Cara membaca koefisien secara hati-hati..... 418
Catatan penting..... 419
Contoh narasi interpretasi..... 419
25.10 Source code Python lengkap: mini project..... 420
25.11 Apa yang sebaiknya ditulis di laporan atau notebook..... 423
Contoh narasi singkat..... 423
25.12 Kesalahan umum pemula..... 424
25.13 Refleksi: mini project regresi adalah jembatan dari belajar ke bekerja..... 424
Penutup Bab..... 425
Bab 26. Mini Project 4 — Klasifikasi End-to-End..... 426
26.1 Tujuan mini project..... 426
26.2 Studi kasus: prediksi kelulusan mahasiswa..... 427
Skenario..... 427
Fitur yang tersedia..... 427
Target..... 427
Kenapa kasus ini cocok..... 427
26.3 Menyiapkan dataset..... 427

Masalah yang sengaja disisipkan....428
26.4 Langkah 1 — Load dataset....429
Apa yang harus dilihat....429
26.5 Langkah 2 — Preprocessing....429
Source code Python lengkap....430
Kenapa preprocessing ini penting....431
26.6 Langkah 3 — Menentukan fitur dan target....431
Fitur yang dipakai....431
Target....431
Catatan....431
Source code Python lengkap....431
26.7 Langkah 4 — Training beberapa model....432
Kenapa tiga model ini....432
Source code Python lengkap....432
26.8 Langkah 5 — Bandingkan hasil model....433
Source code Python lengkap....434
Kenapa F1-score penting....435
26.9 Langkah 6 — Pilih model terbaik....435
Cara memilih model secara sehat....435
Contoh narasi pemilihan model....435
26.10 Source code Python lengkap: mini project terpadu....436
26.11 Apa yang sebaiknya ditulis di laporan atau notebook....438
Contoh narasi singkat....439
26.12 Kesalahan umum pemula....439
26.13 Refleksi: mini project klasifikasi adalah latihan berpikir seperti praktisi....440
Penutup Bab....440
Bab 27. Mini Project 5 — Clustering End-to-End....441
27.1 Tujuan mini project....441
27.2 Studi kasus: segmentasi pelanggan....442
27.3 Menyiapkan dataset pelanggan....442
27.4 Langkah 1 — Load dataset....443
27.5 Langkah 2 — Cleaning dan persiapan fitur....444
Tahapan yang akan dilakukan....444
27.6 Langkah 3 — Visualisasi awal data pelanggan....445
Apa yang harus dilihat....445
27.7 Langkah 4 — Mengapa scaling penting....446
27.8 Langkah 5 — Menentukan jumlah cluster....446
Cara membaca Elbow Method....447
27.9 Langkah 6 — Menjalankan K-Means....447
27.10 Langkah 7 — Visualisasi cluster....448
Apa yang harus dibaca....449
27.11 Langkah 8 — Interpretasi cluster....449
Cara membaca hasilnya....450
27.12 Langkah 9 — Insight bisnis sederhana....450

Contoh insight bisnis sederhana..... 450
Contoh narasi singkat.....451
27.13 Source code Python lengkap: mini project terpadu.....451
27.14 Apa yang sebaiknya ditulis di laporan atau notebook..... 454
Contoh narasi singkat.....454
27.15 Kesalahan umum pemula.....454
27.16 Refleksi: clustering yang baik mengubah data tanpa label menjadi keputusan yang lebih cerdas..... 455
Penutup Bab.....455
Bagian V — Penutup dan Horizon Lanjutan..... 456
Bab 28. Dari Notebook ke Portofolio.....456
28.1 Mengapa portofolio penting.....456
28.2 Dari file latihan menjadi karya yang layak ditunjukkan.....457
28.3 Menyimpan notebook yang rapi..... 458
A. Ciri notebook yang rapi..... 458
B. Struktur notebook yang disarankan..... 458
C. Contoh judul notebook yang baik.....459
D. Hindari notebook yang terlalu “ramai”.....459
E. Contoh susunan Markdown notebook.....459
28.4 Langkah praktis merapikan notebook..... 460
Langkah 1 — pilih satu notebook yang paling layak..... 460
Langkah 2 — hapus sel yang tidak perlu..... 460
Langkah 3 — tambahkan Markdown.....460
Langkah 4 — jalankan notebook dari atas ke bawah.....460
Langkah 5 — cek hasil visualisasi.....460
Langkah 6 — tambahkan kesimpulan..... 460
28.5 Menulis README GitHub..... 461
A. Fungsi README.....461
B. Struktur README yang disarankan.....461
C. Contoh template README GitHub.....462
D. Tips agar README enak dibaca..... 463
28.6 Memilih project portofolio pertama..... 463
A. Karakter proyek pertama yang baik..... 463
B. Proyek pertama yang sangat cocok.....463
C. Pilih proyek yang “punya cerita”.....464
28.7 Contoh tiga proyek portofolio awal yang “menjual”..... 464
1. Proyek EDA.....464
2. Proyek supervised learning..... 464
3. Proyek unsupervised learning..... 465
28.8 Dokumentasi hasil analisis.....465
A. Apa yang perlu didokumentasikan.....465
B. Dokumentasi bukan hanya untuk orang lain.....465
C. Contoh format dokumentasi singkat..... 466
28.9 Struktur repo GitHub yang rapi.....467

Struktur folder sederhana yang disarankan....467
Penjelasan singkat.... 467
Contoh requirements.txt.... 467
28.10 Langkah praktis mengubah proyek menjadi portofolio GitHub....468
Langkah 1 — pilih satu proyek terbaik.... 468
Langkah 2 — rapikan notebook....468
Langkah 3 — siapkan folder repo....468
Langkah 4 — tulis README.... 468
Langkah 5 — tambahkan visualisasi terbaik.... 468
Langkah 6 — unggah ke GitHub.... 468
Langkah 7 — cek dari sudut pandang pembaca baru....468
28.11 Contoh isi README untuk proyek portofolio pertama....469
28.12 Peran dokumentasi visual.... 470
28.13 Apa yang membuat proyek portofolio terlihat “menjual”....470
28.14 Kesalahan umum pemula saat membangun portofolio.... 471
28.15 Refleksi: portofolio adalah bukti bahwa Anda bisa menyelesaikan sesuatu.. 471
Penutup Bab....472
Bab 29. Langkah Berikutnya setelah Buku Ini....473
29.1 Setelah fondasi ML, ke mana sebaiknya melangkah....473
29.2 Deep learning dasar.... 474
A. Kenapa deep learning menjadi langkah lanjutan alami....474
B. Apa yang sebaiknya dipelajari dulu....474
C. Kapan mulai serius ke deep learning....475
D. Sikap yang sehat.... 475
29.3 NLP dan LLM.... 475
A. Kenapa NLP dan LLM penting....475
B. Apa yang sebaiknya dipelajari dulu....475
C. Apa itu LLM secara praktis.... 476
D. Tantangan penting di NLP/LLM....476
29.4 AI coding assistant.... 476
A. Mengapa ini sangat relevan....476
B. Cara memakai AI coding assistant dengan sehat....477
C. Keterampilan penting: menulis prompt yang baik....477
29.5 Deployment sederhana.... 478
A. Kenapa deployment penting.... 478
B. Bentuk deployment paling cocok untuk pemula....478
C. Kenapa Streamlit atau Gradio....478
D. Langkah minimal deployment.... 478
29.6 Jalan belajar yang sehat: dari ML ke AI systems....479
29.7 Roadmap belajar 6 bulan.... 479
Bulan 1 — Menguatkan fondasi dengan proyek ulang....479
Bulan 2 — Pendalaman supervised learning.... 480
Bulan 3 — Pendalaman unsupervised learning + visualisasi....480
Bulan 4 — Masuk ke deep learning dasar....480

Bulan 5 — Masuk ke NLP dan LLM..... 480
Bulan 6 — Deployment ringan + AI coding assistant..... 481
29.8 Cara memilih jalur spesialisasi setelah 6 bulan.....481
Jika suka angka, tabel, dan prediksi bisnis..... 481
Jika suka teks dan bahasa..... 481
Jika suka gambar, audio, atau sinyal.....482
Jika suka membangun tool.....482
Jika suka sistem yang lebih otonom.....482
29.9 Apa yang sebaiknya tetap dijaga saat belajar lanjutan..... 482
29.10 Sumber latihan dan komunitas.....483
29.11 Kesalahan umum setelah selesai belajar dasar..... 483
29.12 Refleksi: setelah buku ini, yang paling penting adalah ritme.....484
Penutup Bab.....484
Bab 30. Penutup — Python sebagai Jalan Masuk ke Dunia AI..... 485
Lampiran A. Cheat Sheet Perintah Ubuntu 26.04 untuk Python.....487
A.1 Update sistem dan instal Python.....487
A.2 Cek versi Python dan pip..... 487
A.3 Membuat folder proyek..... 487
A.4 Membuat virtual environment.....488
A.5 Mengaktifkan virtual environment..... 488
A.6 Menonaktifkan virtual environment..... 488
A.7 Menginstal package Python.....488
A.8 Menyimpan daftar package.....488
A.9 Menjalankan Python interaktif.....488
A.10 Menjalankan file Python.....489
A.11 Menjalankan Jupyter Notebook.....489
A.12 Perintah yang sangat sering dipakai.....489
A.13 Cheatsheet singkat Ubuntu 26.04 untuk Python.....490
Lampiran B. Cheat Sheet Python Dasar..... 491
B.1 Tipe data dasar..... 491
B.2 Struktur data penting.....491
B.3 Percabangan if.....492
B.4 Loop for.....492
B.5 Loop while.....492
B.6 Fungsi..... 492
B.7 Input pengguna..... 492
B.8 Membaca file teks..... 492
B.9 Menulis file teks.....493
B.10 Membaca CSV dengan modul standar..... 493
B.11 Membaca JSON..... 493
B.12 Penanganan error dasar..... 493
B.13 Cheat Sheet super singkat Python dasar.....494
Lampiran C. Cheat Sheet Data Science..... 495
C.1 NumPy..... 495

Import..... 495
Membuat array..... 495
Array 2 dimensi.....495
Bentuk array..... 495
Membuat array khusus.....495
Statistik dasar.....495
Reshape..... 495
C.2 Pandas.....496
Import..... 496
Membaca CSV..... 496
Lima baris pertama.....496
Info data.....496
Statistik ringkas..... 496
Pilih kolom..... 496
Pilih beberapa kolom.....496
Filter data.....496
Urutkan data.....496
Groupby.....496
Missing value.....496
Simpan CSV.....496
C.3 Matplotlib..... 497
Import..... 497
Line plot.....497
Bar plot..... 497
Histogram..... 497
Scatter plot..... 497
C.4 scikit-learn.....498
Import umum..... 498
Train-test split.....498
Linear Regression..... 498
Logistic Regression.....498
KMeans..... 498
Scaling.....498
Lampiran D. Template Code Machine Learning.....499
D.1 Template regresi..... 499
D.2 Template klasifikasi.....500
D.3 Template clustering.....501
Lampiran E. Dataset Gratis untuk Latihan..... 502
E.1 Iris..... 502
E.2 Titanic.....502
E.3 Wine.....502
E.4 Housing.....503
E.5 Customer segmentation.....503
E.6 Dataset synthetic.....503

Referensi.....505
Tentang Penulis.....506

Bagian I — Mulai dari Nol di Ubuntu 26.04

Bab 1. Python, AI, dan Jalan Belajar yang Masuk Akal

Bab ini menjadi pintu masuk paling penting bagi pembaca. Sebelum membahas *syntax*, *library*, atau model AI, pembaca perlu lebih dulu memahami **mengapa Python dipilih**, **bagaimana jalur belajar yang realistis**, dan **cara memakai buku ini secara efektif**. Di outline naskah utama, Bab 1 memang ditempatkan sebagai fondasi agar pembaca tidak sekadar “ikut tutorial”, tetapi memiliki gambaran utuh tentang perjalanan belajar dari nol menuju kemampuan membuat solusi AI/ML sederhana yang bermanfaat. Naskah utama juga menegaskan bahwa target buku ini adalah **pemula total, mahasiswa, guru, profesional non-IT, dan analis pemula** yang ingin masuk ke AI/ML secara bertahap dalam rentang **4–6 bulan belajar konsisten**.

Secara umum, jalur yang diambil buku ini sangat masuk akal: pembaca tidak langsung diajak “melompat” ke model yang rumit, tetapi dibimbing dari **Python dasar**, lanjut ke **pengolahan data**, lalu ke **statistik praktis**, kemudian masuk ke **machine learning**, dan akhirnya diperkenalkan pada horizon **AI modern** seperti *deep learning*, *LLM*, *AI coding assistant*, dan *deployment* ringan. Struktur seperti ini juga konsisten dengan materi pada buku referensi yang menempatkan Python sebagai dasar, dilanjutkan dengan *data manipulation*, visualisasi, *EDA*, statistik, dan *machine learning workflow*.

Mengapa Python menjadi bahasa utama untuk data science, AI, dan otomasi

Python menjadi bahasa utama di dunia *data science*, AI, dan otomasi bukan karena paling tua atau paling cepat, tetapi karena **paling seimbang** antara kemudahan belajar, keterbacaan kode, dan kekuatan ekosistem. Naskah utama secara jelas menempatkan Python sebagai bahasa yang **mudah dibaca, fleksibel, dan kaya *library***, sehingga sangat cocok untuk pembaca pemula yang ingin cepat produktif. Buku referensi data analysis juga menekankan bahwa Python dikenal sebagai bahasa yang **beginner-friendly**, dengan *syntax* yang sederhana dan mudah dipahami.

Alasan pertama mengapa Python begitu dominan adalah karena **kode Python relatif mudah dibaca seperti bahasa manusia**. Dibanding banyak bahasa lain yang penuh simbol, kurung, dan struktur yang kaku, Python mendorong gaya penulisan yang lebih bersih. Ini sangat membantu mahasiswa dan praktisi yang baru mulai belajar pemrograman. Mereka tidak perlu habis tenaga hanya untuk memahami bentuk penulisan kode; energi bisa langsung diarahkan untuk memahami logika, data, dan pemecahan masalah. Buku referensi juga menegaskan bahwa *indentation* dan *readability* adalah kekuatan utama Python untuk pembelajaran awal.

Alasan kedua adalah **ekosistem *library* Python sangat matang**. Untuk komputasi numerik ada *NumPy*, untuk pengolahan tabel data ada *Pandas*, untuk visualisasi ada *Matplotlib* dan *Seaborn*, dan untuk *machine learning* ada *scikit-learn*. Di referensi data analysis, fondasi ini tampak sangat jelas: pembelajaran bergerak dari Python dasar, ke NumPy, Pandas, *data*

cleaning, visualisasi, *EDA*, lalu *machine learning*. Ini menunjukkan bahwa Python bukan hanya bahasa pemrograman umum, tetapi benar-benar telah menjadi **bahasa kerja** untuk dunia data.

Alasan ketiga adalah Python sangat kuat untuk **otomasi pekerjaan nyata**. Seorang praktisi tidak selalu membutuhkan AI tingkat tinggi di hari pertama. Sering kali yang dibutuhkan adalah mengotomasi pekerjaan berulang: membersihkan file CSV, menggabungkan data, membuat ringkasan laporan, memproses data transaksi, atau menyiapkan dataset untuk analisis lanjutan. Karena itu, Python memberi kemenangan awal yang cepat. Pembaca bisa merasakan manfaat hanya dari beberapa puluh baris kode. Hal ini penting untuk menjaga motivasi belajar, karena hasilnya terasa nyata sejak awal. Arah seperti ini juga sejalan dengan semangat buku-buku referensi yang menekankan proyek praktis dan pengolahan data nyata, bukan teori kosong.

Alasan keempat adalah Python menjadi jembatan yang mulus menuju AI. Dalam banyak kasus, model AI modern tetap membutuhkan fondasi yang sama: membaca data, membersihkan data, mengubah data menjadi fitur, melatih model, mengevaluasi model, lalu menyimpan atau menampilkan hasil. Dengan kata lain, **belajar Python bukan langkah sampingan**, tetapi memang **jalan masuk utama** ke dunia AI/ML. Referensi machine learning juga menekankan pentingnya pemahaman tentang *training set*, *testing set*, evaluasi, kualitas data, serta perbedaan *supervised* dan *unsupervised learning* sebelum masuk lebih jauh ke sistem AI yang lebih canggih.

Agar lebih mudah dipahami, bayangkan Python seperti **bahasa pengantar universal** di laboratorium data. Jika data adalah bahan mentah, statistik adalah cara berpikir, dan AI adalah mesin cerdasnya, maka Python adalah alat kerja yang menghubungkan semuanya. Dengan Python, pembaca tidak hanya belajar menulis program, tetapi belajar **berdialog dengan data**.

Peta perjalanan belajar: Python → data → statistik → ML → AI lanjutan

Salah satu kesalahan umum pemula adalah ingin langsung belajar AI yang “hebat” tanpa fondasi yang cukup. Akibatnya, mereka bisa menyalin kode dari internet, tetapi tidak memahami mengapa kode itu bekerja, kapan ia gagal, dan bagaimana memperbaikinya. Karena itu, buku ini memakai jalur belajar yang lebih sehat: **Python dasar terlebih dahulu, lalu data, statistik, machine learning, dan terakhir AI lanjutan**. Jalur ini ditegaskan berulang dalam outline naskah utama dan juga sangat konsisten dengan struktur buku-buku referensi pendamping.

Tahap pertama adalah **Python dasar**. Di tahap ini pembaca fokus pada hal-hal mendasar: variabel, tipe data, operasi logika, *list*, *dictionary*, *loop*, fungsi, file, CSV, dan JSON. Tujuan tahap ini bukan menjadikan pembaca programmer profesional, tetapi membuat pembaca **nyaman berpikir dengan kode**. Mereka belajar bahwa komputer bisa diminta melakukan tugas berulang, menghitung, menyimpan data, dan mengeksekusi logika secara konsisten. Naskah utama setelah Bab 1 memang langsung mengalir ke penyiapan lingkungan belajar, Python dasar, struktur data, kontrol alur program, fungsi, dan file handling.

Tahap kedua adalah **belajar bekerja dengan data**. Setelah bisa menulis kode dasar, pembaca mulai masuk ke dunia *NumPy* dan *Pandas*. Pada tahap ini, data tidak lagi dipandang sebagai angka acak, tetapi sebagai sesuatu yang bisa dibaca, difilter, dikelompokkan, dibersihkan, dan dianalisis. Referensi data analysis menunjukkan urutan yang sangat alami: setelah Python dasar, pembaca masuk ke *data manipulation*, *cleaning*, dan visualisasi. Ini menunjukkan bahwa dalam praktik nyata, kemampuan mengolah data justru lebih sering dipakai daripada algoritma AI yang canggih.

Tahap ketiga adalah **statistik praktis**. Ini bagian yang sangat penting. Banyak pemula ingin melewati statistik karena dianggap sulit atau terlalu matematis. Padahal, naskah utama secara tegas memilih pendekatan bahwa statistik harus diajarkan sebagai **alat berpikir praktis**, bukan sekadar kumpulan rumus. Pembaca perlu memahami rata-rata, median, variasi, distribusi, outlier, korelasi, dan regresi, karena semua itu membantu menjawab pertanyaan: “Apa yang sebenarnya terjadi pada data ini?” Tanpa statistik, pembaca hanya akan menekan tombol *run* tanpa tahu apa yang sedang dihitung.

Tahap keempat adalah **machine learning**. Setelah memahami data dan statistik, barulah pembaca masuk ke model. Pada tahap ini pembaca belajar perbedaan **regresi** dan **klasifikasi**, perbedaan **supervised** dan **unsupervised learning**, pentingnya **train-test split**, serta cara mengevaluasi model dengan metrik yang tepat. Naskah utama memosisikan bagian ini sebagai inti implementasi, meliputi *workflow ML*, klasifikasi dasar, evaluasi model, regresi, model lanjutan ringan, dan *unsupervised learning* seperti *clustering* dan PCA. Ini juga didukung oleh referensi lain yang membahas *linear models*, *classic classifiers*, *supervised learning*, dan *unsupervised learning*.

Tahap kelima adalah **AI lanjutan**. Di sini pembaca mulai mengenal cakrawala yang lebih luas: *deep learning*, pemrosesan teks, *LLM*, *AI coding assistant*, atau *deployment* ringan. Namun naskah utama juga mengingatkan bahwa bagian ini harus menjadi **pengantar horizon lanjutan**, bukan inti utama, agar fondasi Python, data, dan ML tidak dikorbankan. Ini keputusan pedagogis yang sangat tepat. Pembaca pemula lebih baik paham dasar dengan kuat daripada cepat menyentuh topik modern tetapi rapuh secara konsep.

Agar peta ini lebih membumi, berikut gambaran sederhana per tahap:

- **Tahap 1: Python dasar**
  Belajar menulis kode, memakai variabel, fungsi, dan file.
- **Tahap 2: Data**
  Belajar membaca CSV, membuat tabel, membersihkan data, dan visualisasi.
- **Tahap 3: Statistik**
  Belajar memahami pola, hubungan, sebaran, dan kualitas data.
- **Tahap 4: ML**
  Belajar membuat model prediksi dan segmentasi sederhana.
- **Tahap 5: AI lanjutan**
  Belajar arah masa depan setelah fondasi cukup kuat.

Urutan ini membuat pembelajaran terasa **masuk akal, bertahap, dan tidak menakutkan**. Pembaca tidak disuruh mendaki gunung dalam satu lompatan. Mereka diajak menaiki tangga, satu anak tangga demi satu anak tangga.

Apa yang realistis dicapai pemula dalam 4–6 bulan

Salah satu hal paling menenangkan bagi pembaca adalah mengetahui target yang realistis. Naskah utama secara eksplisit menyebut bahwa buku ini ditujukan untuk pembaca yang ingin masuk ke AI/ML dalam **4–6 bulan belajar konsisten**, bukan dalam dua hari, bukan pula dalam janji palsu "jadi ahli AI instan". Target akhirnya juga dirumuskan dengan jelas: pembaca diharapkan mampu memahami Python dasar untuk analisis data, membersihkan dan mengeksplorasi dataset nyata, membangun model klasifikasi dan regresi sederhana-menengah, membaca metrik evaluasi model dengan benar, membuat mini portofolio berbasis notebook, dan mengenal arah lanjutan.

Dalam **bulan pertama**, target yang realistis adalah pembaca mulai nyaman dengan Python dasar dan lingkungan kerja. Mereka sudah mampu menjalankan file `.py`, mencoba notebook, menggunakan `pip`, memahami variabel, kondisi, perulangan, fungsi, dan membaca file sederhana. Pada tahap ini, keberhasilan bukan diukur dari "banyaknya *library* yang dipasang", tetapi dari rasa percaya diri bahwa kode yang ditulis sendiri bisa berjalan dan menghasilkan sesuatu.

Dalam **bulan kedua**, pembaca mulai masuk ke pengolahan data. Mereka belajar menggunakan *Pandas* untuk membaca CSV, memilih kolom, memfilter data, mengurutkan, mengelompokkan, dan membersihkan data yang belum rapi. Di tahap ini, pembaca mulai bisa mengerjakan studi kasus sederhana seperti data penjualan, data siswa, data transaksi, atau dataset publik ringan. Referensi data analysis memang menekankan bahwa setelah Python dasar, kemampuan manipulasi data adalah langkah berikutnya yang paling penting.

Dalam **bulan ketiga**, pembaca mulai belajar statistik praktis dan visualisasi. Mereka seharusnya sudah mampu membaca distribusi data, membedakan mean dan median, melihat outlier, menghitung korelasi sederhana, serta membuat grafik dasar seperti *line plot*, *bar plot*, *histogram*, *boxplot*, dan *scatter plot*. Ini adalah tahap ketika data mulai "berbicara". Pembaca bukan sekadar melihat angka, tetapi mulai melihat pola. Outline naskah utama menempatkan fase ini secara jelas melalui bab visualisasi dan statistik praktis.

Dalam **bulan keempat**, pembaca mulai masuk ke *machine learning* dasar. Mereka bisa mulai dengan *Linear Regression* untuk prediksi angka, kemudian *Logistic Regression*, KNN, *Decision Tree*, atau *Random Forest* untuk klasifikasi sederhana. Pada fase ini, pembaca juga mulai belajar *train_test_split*, evaluasi model, dan perbedaan antara model yang bagus di data latih dan model yang benar-benar berguna di data baru. Referensi pendukung juga menunjukkan bahwa topik seperti *linear models*, klasifikasi, *supervised vs unsupervised*, dan evaluasi metrik merupakan inti pembelajaran tahap menengah awal.

Dalam **bulan kelima sampai keenam**, pembaca yang konsisten biasanya sudah cukup siap untuk membuat **mini project**. Misalnya, analisis data penjualan, prediksi harga rumah sederhana, klasifikasi pelanggan, segmentasi pelanggan dengan *K-Means*, atau notebook ringkas yang menunjukkan proses dari *loading data* sampai interpretasi hasil. Inilah tahap yang sangat penting karena pembaca mulai mengubah

pengetahuan menjadi **portofolio**. Referensi juga merekomendasikan *practice projects*, dataset publik, dan keterlibatan dalam komunitas agar kemampuan terus bertumbuh.

Supaya lebih konkret, berikut capaian yang **realistis** untuk pemula dalam 4–6 bulan bila belajar konsisten:

- **Mampu menulis dan membaca kode Python dasar** tanpa terlalu bergantung pada salin-tempel.
- **Mampu membaca dan membersihkan dataset sederhana** menggunakan *Pandas*.
- **Mampu membuat visualisasi dasar** dan menjelaskan insight awal dari data.
- **Mampu memahami statistik dasar** yang relevan untuk analisis.
- **Mampu melatih model regresi dan klasifikasi sederhana** dengan *scikit-learn*.
- **Mampu membaca metrik evaluasi** seperti MAE, RMSE, accuracy, precision, recall, dan F1-score pada level praktis.
- **Mampu membuat 2–3 mini project portofolio** dalam bentuk notebook yang rapi.

Yang penting dipahami, target realistis itu bukan berarti kecil. Justru sebaliknya, untuk seorang pemula, kemampuan di atas sudah sangat bernilai. Dengan bekal tersebut, pembaca sudah bisa melanjutkan ke *deep learning*, NLP, *LLM*, analitik bisnis, atau otomasi kerja yang lebih kompleks.

Cara memakai buku ini: praktik, eksperimen, mini project

Buku ini tidak dirancang untuk dibaca seperti novel yang selesai sekali duduk. Buku ini lebih tepat dipakai sebagai **buku kerja**. Naskah utama secara tegas menekankan bahwa pembelajaran harus dilakukan dengan **kode, eksperimen, dan mini project**. Pendekatan ini sangat tepat untuk topik Python dan AI, karena keterampilan seperti ini tidak cukup hanya dipahami di kepala; ia harus dilatih di tangan.

Cara pertama memakai buku ini adalah **jangan hanya membaca, tetapi langsung mencoba setiap contoh**. Saat melihat potongan kode, pembaca perlu mengetikkannya sendiri, menjalankannya, lalu melihat hasilnya. Mengetik ulang jauh lebih baik daripada hanya menyalin-tempel, karena proses ini membantu otak mengenali pola penulisan, struktur logika, dan sumber kesalahan umum.

Cara kedua adalah **membiasakan eksperimen kecil**. Setelah contoh berjalan, ubahlah sedikit. Ganti angka, nama variabel, kondisi, ukuran data, atau jenis grafik. Dari sini pembaca akan mulai menyadari bahwa belajar Python dan AI bukan soal menghafal, tetapi soal memahami efek dari perubahan yang dilakukan. Eksperimen semacam ini membuat pembelajaran lebih hidup, lebih menantang, dan lebih menyenangkan.

Cara ketiga adalah **membuat catatan kesalahan**. Dalam praktik Python, *error* bukan musuh, tetapi bagian dari proses belajar. Bahkan dalam outline buku utama, pembahasan tentang membaca *error* dan *debugging* memang ditempatkan sebagai materi penting. Jadi, ketika program gagal jalan, pembaca tidak perlu panik. Justru di situlah mereka sedang belajar membaca bahasa mesin.

Cara keempat adalah **menyelesaikan mini project secara bertahap**. Mini project penting karena di sanalah semua bagian bertemu: Python dasar, data, statistik, visualisasi, dan model. Pembaca sebaiknya tidak menunggu “jago dulu” untuk membuat project. Sebaliknya, project kecil justru membantu pembaca bertumbuh lebih cepat. Naskah utama memang sengaja menempatkan mini project di beberapa tahap agar pembaca terbiasa menyatukan konsep menjadi hasil nyata.

Cara kelima adalah **mengulang topik penting, bukan sekadar lanjut halaman**. Dalam belajar Python dan data, pemahaman sering tumbuh berlapis. Sesuatu yang terasa membingungkan hari ini bisa menjadi sangat jelas dua minggu kemudian setelah mencoba beberapa latihan. Karena itu, pembaca tidak perlu malu untuk kembali ke bab sebelumnya. Proses maju-mundur seperti ini sangat normal.

Berikut langkah praktis memakai buku ini agar hasil belajar lebih terasa:

- **Langkah 1: Baca ringkas tujuan bab**
  Pahami dulu apa yang ingin dicapai pada bab tersebut.
- **Langkah 2: Jalankan semua contoh**
  Ketik ulang, jangan hanya dibaca.
- **Langkah 3: Modifikasi contoh**
  Ubah input, tambah kondisi, atau ganti dataset kecil.
- **Langkah 4: Catat error dan solusi**
  Buat jurnal belajar singkat agar pola kesalahan mudah dikenali.
- **Langkah 5: Kerjakan latihan mini**
  Walau sederhana, ini bagian yang paling membentuk kebiasaan.
- **Langkah 6: Selesaikan mini project**
  Anggap mini project sebagai bukti bahwa ilmu mulai berubah menjadi keterampilan.
- **Langkah 7: Simpan hasil di notebook atau GitHub**
  Ini akan sangat membantu saat membangun portofolio di tahap berikutnya.

Penutup Bab

Bab ini seharusnya memberi rasa tenang sekaligus semangat. **Python bukan gerbang yang menakutkan**, melainkan jalan masuk yang logis ke dunia data, AI, dan otomasi. Pembaca tidak perlu menjadi ahli matematika atau programmer senior untuk mulai. Yang dibutuhkan adalah **kemauan mencoba, keberanian salah, dan konsistensi belajar sedikit demi sedikit**. Naskah utama memang merancang buku ini agar pembaca bisa berkembang dari nol menjadi seseorang yang mampu memahami data, membangun model sederhana, dan menyiapkan portofolio awal dalam waktu 4–6 bulan belajar konsisten.

Dengan semangat itu, Bab 1 bukan hanya pengantar, tetapi **kompas belajar**. Setelah memahami alasan memilih Python, peta perjalanan belajar, target realistis, dan cara memakai buku ini, pembaca akan masuk ke bab-bab berikutnya dengan lebih siap, lebih percaya diri, dan lebih menikmati prosesnya. Di sinilah perjalanan dimulai: bukan untuk menjadi “sekadar pengguna AI”, tetapi untuk menjadi **orang yang mengerti, mencoba, membangun, dan terus bertumbuh bersama teknologi**.

Bab 2. Menyiapkan Ecosystem Python

Bab ini sengaja dibuat **singkat, padat, dan langsung pakai**. Tujuannya bukan membuat pembaca menjadi ahli instalasi, tetapi memastikan pembaca segera punya lingkungan kerja yang cukup nyaman untuk mulai belajar Python, analisis data, statistik, dan *machine learning*. Di outline naskah utama, Bab 2 memang difokuskan pada **instal Python, memilih editor seperti VS Code, Jupyter Notebook, dan Google Colab, memahami `pip` dan `venv`, menata folder proyek, menjalankan file `.py` dan `.ipynb`, serta menghindari error pemula**. Arah ini juga sejalan dengan dokumen referensi yang menekankan bahwa fondasi awal *data work* adalah *environment setup*, Jupyter, editor, dan manajemen *package*, bukan setup yang terlalu panjang atau rumit.

Secara praktis, untuk pembaca buku ini ada **dua jalur *ecosystem* yang sangat masuk akal**. Jalur pertama adalah **Ubuntu 26.04**, cocok bagi pembaca yang ingin belajar lebih serius, lebih mandiri, dan lebih dekat dengan lingkungan kerja teknis yang umum dipakai praktisi. Jalur kedua adalah **Google Colab**, cocok bagi pembaca yang ingin cepat mencoba tanpa instalasi lokal, terutama bila komputer yang dipakai sederhana atau masih belum siap untuk setup penuh. Dengan dua pilihan ini, pembaca bisa mulai dari yang paling nyaman, lalu naik kelas secara bertahap. Ini penting karena tujuan buku bukan menguji kesabaran pembaca saat instalasi, tetapi **membawa pembaca secepat mungkin ke praktik yang produktif**.

2.1. Dua pilihan ecosystem: Ubuntu 26.04 atau Google Colab

Sebelum masuk ke langkah teknis, pembaca perlu memahami bahwa **tidak ada satu jalur yang wajib untuk semua orang**. Yang penting adalah memilih jalur yang paling membuat Anda bisa mulai belajar hari ini.

1. Ubuntu 26.04: jalur yang lebih kuat dan rapi

Ubuntu 26.04 cocok untuk pembaca yang ingin membangun kebiasaan kerja profesional sejak awal. Dengan Ubuntu, pembaca belajar memakai terminal, mengelola *virtual environment*, memasang *package* dengan rapi, menjalankan script `.py`, dan bekerja dengan notebook `.ipynb` secara lokal. Jalur ini sangat cocok untuk mahasiswa teknik, praktisi IT, analis data pemula, atau siapa pun yang ingin memahami proses dari bawah secara lebih utuh. Outline naskah utama memang menempatkan `pip`, `venv`, VS Code, dan Jupyter sebagai fondasi lingkungan belajar yang nyaman.

Kelebihan Ubuntu 26.04:

- **Lebih dekat ke lingkungan kerja nyata**
- **Bisa dipakai offline** setelah instalasi selesai
- **Kontrol penuh** atas versi Python, *library*, dan file proyek
- Sangat baik untuk membangun kebiasaan kerja yang rapi

Kapan memilih Ubuntu 26.04?

- Anda ingin serius belajar jangka panjang
- Anda ingin memahami terminal dan struktur proyek
- Anda ingin menyiapkan dasar yang kuat untuk project AI/ML yang lebih besar

2. Google Colab: jalur cepat, ringan, dan praktis

Google Colab cocok untuk pembaca yang ingin **langsung mulai menulis kode tanpa repot instalasi lokal**. Karena berjalan di browser, pembaca hanya perlu akun Google dan koneksi internet. Colab sangat baik untuk latihan awal, eksperimen cepat, atau ketika laptop belum siap dipasang banyak *library*. Di outline utama, Google Colab sudah disebut sebagai salah satu pilihan editor/lingkungan belajar yang valid untuk pemula.

Kelebihan Google Colab:

- **Tidak perlu instalasi lokal**
- Bisa langsung dipakai dari browser
- Cocok untuk latihan cepat dan eksplorasi awal
- Mudah berbagi notebook

Kapan memilih Google Colab?

- Anda ingin mulai belajar hari ini tanpa setup teknis
- Spesifikasi laptop terbatas
- Anda ingin fokus dulu ke kode dan konsep

Rekomendasi praktis untuk pembaca

Untuk buku ini, strategi yang paling sehat adalah:

- **Mulai dari Colab** bila ingin cepat mencoba
- **Naik ke Ubuntu 26.04** bila ingin belajar lebih serius dan terstruktur

Dengan cara ini, pembaca tidak terjebak pada dilema “harus sempurna dulu baru mulai”. Yang penting adalah mulai. Setelah nyaman, barulah pindah ke lingkungan yang lebih kuat.

2.2. Jalur 1 — Menyiapkan Python di Ubuntu 26.04

Untuk pembaca yang memilih Ubuntu 26.04, berikut langkah yang sederhana dan aman. Fokusnya adalah menyiapkan Python, `pip`, `venv`, VS Code, Jupyter Notebook, dan *library* inti seperti `numpy`, `pandas`, `matplotlib`, dan `scikit-learn`. Referensi pendamping juga menegaskan bahwa Python environment yang baik untuk analisis data biasanya mencakup Jupyter, editor seperti VS Code/PyCharm, serta manajemen *package* dengan `pip` dan *virtual environment*.

1. Instalasi Python di Ubuntu 26.04

Langkah pertama adalah memastikan Python 3, `pip`, dan `venv` tersedia.

Buka terminal, lalu jalankan:

```
sudo apt update
sudo apt install -y python3 python3-pip python3-venv
```

Setelah itu, cek versinya:

```
python3 --version
pip3 --version
```

Contoh keluaran:

```
Python 3.12.3
pip 24.0 from /usr/lib/python3/dist-packages/pip (python 3.12)
```

Langkah verifikasi seperti ini penting karena banyak masalah pemula sebenarnya hanya soal Python belum terpasang dengan benar atau `pip` belum tersedia. Buku pengantar Python dari Abdullah juga menekankan langkah dasar yang sama: instal Python, lalu cek versi dengan perintah `python --version` untuk memastikan instalasi berhasil.

2. Memahami `python3`, `pip`, dan `venv`

Setelah Python terpasang, pembaca perlu memahami peran tiga komponen inti ini.

- **`python3`**: interpreter untuk menjalankan kode Python
- **`pip`** / **`pip3`**: alat untuk memasang *library*
- **`venv`**: alat untuk membuat lingkungan proyek yang terpisah

Referensi Jain menekankan bahwa `pip` adalah *package manager* standar Python, dan praktik terbaiknya adalah memakai *virtual environment* serta menyimpan dependensi di `requirements.txt` menggunakan `pip freeze > requirements.txt`.

3. Membuat proyek pertama dan *virtual environment*

Misalnya kita ingin membuat folder proyek bernama `python-ai-pemula`.

```
mkdir -p ~/projek/python-ai-pemula
cd ~/projek/python-ai-pemula
python3 -m venv .venv
source .venv/bin/activate
```

Jika aktif, terminal biasanya berubah menjadi seperti ini:

```
(.venv) user@ubuntu:~/projek/python-ai-pemula$
```

Sekarang perintah `python` dan `pip` akan mengarah ke lingkungan proyek ini. Ini adalah kebiasaan yang sangat baik sejak awal, karena setiap proyek akan punya dependensi sendiri dan tidak saling bertabrakan. Referensi Morgan juga menegaskan pentingnya menyiapkan *virtual environment* saat membangun aplikasi Python pertama.

4. Memperbarui pip

Setelah `venv` aktif, jalankan:

```
python -m pip install --upgrade pip
```

Ini bukan wajib setiap saat, tetapi baik dilakukan di awal agar instalasi paket berikutnya lebih lancar.

2.3. VS Code dan Jupyter Notebook di Ubuntu 26.04

Outline utama secara jelas memasukkan **VS Code** dan **Jupyter Notebook** sebagai editor utama. Ini keputusan yang tepat, karena keduanya saling melengkapi. Jupyter sangat ideal untuk eksplorasi data dan pembelajaran interaktif, sedangkan VS Code sangat baik untuk proyek yang mulai lebih rapi dan panjang. Referensi juga menegaskan bahwa Jupyter adalah lingkungan interaktif untuk menulis dan menjalankan kode, sementara VS Code menyediakan *interactive notebooks*, *debugging*, Git integration, dan berbagai *extension* produktivitas.

1. Mengapa Jupyter Notebook penting

Jupyter sangat cocok untuk pemula karena memungkinkan pembaca:

- menulis kode sedikit demi sedikit dalam *cell*
- langsung melihat output di bawah *cell*
- menambahkan catatan dan penjelasan dengan *Markdown*
- menggabungkan kode, tabel, dan grafik dalam satu dokumen

Referensi juga menyebut Jupyter sebagai lingkungan yang sangat baik untuk *data exploration*, visualisasi, dan berbagi hasil.

2. Instal Jupyter Notebook

Pastikan `venv` aktif, lalu jalankan:

```
pip install notebook jupyterlab ipykernel
```

Lalu jalankan:

```
jupyter notebook
```

Menurut referensi, langkah awal memakai Jupyter adalah membuka dari terminal dengan `jupyter notebook`, kemudian membuat notebook baru melalui **New > Python 3**.

3. Mengapa VS Code penting

Untuk proyek yang mulai membesar, VS Code sangat membantu karena menyediakan:

- *code completion*
- *debugging*
- Git integration
- dukungan notebook interaktif melalui extension Jupyter
- dukungan Python melalui extension resmi

Referensi Jain menjelaskan bahwa VS Code adalah editor yang ringan, dapat disesuaikan, mendukung Python melalui *extension*, dan bisa menjalankan notebook langsung dari dalam VS Code.

4. Langkah memakai VS Code

Setelah VS Code terpasang di Ubuntu 26.04:

- buka folder proyek
- instal extension **Python**
- instal extension **Jupyter**
- pilih interpreter yang mengarah ke `.venv`
- buka file `.py` atau `.ipynb`

Hal ini penting karena salah satu error pemula yang sering terjadi adalah **interpreter salah**. Outline utama juga sudah mengingatkan soal error seperti kernel salah dan *package* tidak ditemukan.

2.4. Struktur folder proyek yang rapi

Struktur folder yang rapi akan sangat membantu pembaca saat mulai mengerjakan latihan, mini project, dan portofolio. Sejak awal, biasakan memisahkan data, notebook, script, dan output.

Contoh struktur sederhana:

```
python-ai-pemula/
├── .venv/
├── data/
│   ├── raw/
│   └── processed/
├── notebooks/
├── src/
├── outputs/
├── requirements.txt
└── README.md
```

Penjelasan singkat:

- **`.venv/`**: lingkungan Python proyek
- **`data/raw/`**: data mentah
- **`data/processed/`**: data hasil pembersihan
- **`notebooks/`**: file `.ipynb`
- **`src/`**: file `.py`
- **`outputs/`**: grafik, model, atau hasil ekspor
- **`requirements.txt`**: daftar *library*
- **`README.md`**: catatan ringkas proyek

Buat struktur ini dengan:

```
mkdir -p data/raw data/processed notebooks src outputs
touch requirements.txt README.md
```

Lalu simpan dependensi proyek:

```
pip freeze > requirements.txt
```

Praktik menyimpan dependensi ini juga direkomendasikan dalam referensi sebagai bagian dari *best practice* pengelolaan proyek Python.

2.5. Instal *library* inti: `numpy`, `pandas`, `matplotlib`, `scikit-learn`

Untuk buku ini, pembaca belum perlu memasang terlalu banyak *library*. Cukup beberapa inti yang akan dipakai berulang kali:

- **NumPy** untuk komputasi numerik
- **Pandas** untuk tabel data
- **Matplotlib** untuk grafik
- **scikit-learn** untuk *machine learning*

Referensi Jain secara eksplisit menyebut *scikit-learn* sebagai *machine learning library* untuk klasifikasi, regresi, *clustering*, dan analisis data, sedangkan Jupyter disebut sebagai lingkungan interaktif untuk eksplorasi dan visualisasi.

Instal dengan:

```
pip install numpy pandas matplotlib scikit-learn
```

Atau langsung lengkap dengan Jupyter:

```
pip install numpy pandas matplotlib scikit-learn notebook
jupyterlab ipykernel
```

Sesudah itu, simpan dependensi:

```
pip freeze > requirements.txt
```

Uji cepat instalasi

Buat file `src/cek_library.py`:

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

print("NumPy OK:", np.__version__)
print("Pandas OK:", pd.__version__)
print("Matplotlib OK:", plt.matplotlib.__version__)
print("Scikit-learn OK:", sklearn.__version__)
```

Jalankan:

```
python src/cek_library.py
```

Kalau semua tampil tanpa error, berarti lingkungan kerja sudah siap.

2.5. Menjalankan file `.py`

File `.py` cocok untuk script Python biasa. Ini penting karena banyak latihan dan mini project nantinya akan ditulis sebagai script atau modul sederhana.

Contoh file Python pertama

Buat file `src/hallo.py`:

```
print("Halo, Ubuntu 26.04!")
print("Saya siap belajar Python, data, dan AI.")
```

Jalankan:

```
python src/hallo.py
```

Contoh hasil:

```
Halo, Ubuntu 26.04!
Saya siap belajar Python, data, dan AI.
```

Contoh script kecil dengan pandas

Buat file `src/contoh_data.py`:

```
import pandas as pd

data = {
    "nama": ["Andi", "Budi", "Citra"],
    "nilai": [88, 75, 93]
}

df = pd.DataFrame(data)
print(df)
print("\nRata-rata nilai:", df["nilai"].mean())
```

Jalankan:

```
python src/contoh_data.py
```

Ini contoh sederhana, tetapi sangat bagus untuk menunjukkan bahwa Python bisa langsung dipakai untuk mengolah data nyata dengan cepat.

2.6. Menjalankan notebook `.ipynb`

Notebook `.ipynb` cocok untuk eksplorasi, latihan bertahap, dan dokumentasi belajar. Referensi menjelaskan bahwa notebook memungkinkan pembaca menulis kode, menjalankan *cell*, menggunakan *Markdown*, dan menggabungkan narasi dengan analisis dalam satu dokumen.

Langkah membuat notebook pertama

1. Aktifkan `venv`
2. Jalankan:

```
jupyter notebook
```

3. Di browser, pilih **New > Python 3**
4. Simpan notebook, misalnya `notebooks/perkenalan.ipynb`

Isi *cell* pertama

```
print("Halo dari Jupyter Notebook!")
```

Tekan **Shift + Enter**.

Isi *cell* kedua

```
import pandas as pd

df = pd.DataFrame({
    "bulan": ["Jan", "Feb", "Mar"],
    "penjualan": [120, 150, 170]
})

df
```

Isi *cell* ketiga untuk visualisasi

```
import matplotlib.pyplot as plt

df.plot(x="bulan", y="penjualan", kind="bar")
plt.title("Penjualan Tiga Bulan")
plt.show()
```

Referensi juga menekankan bahwa notebook sangat baik untuk eksplorasi, visualisasi, dan dokumentasi analisis, terutama karena bisa menggabungkan *code cells* dan *Markdown cells*.

2.7. Jalur 2 — Google Colab untuk pembaca yang ingin cepat mulai

Bagi pembaca yang tidak ingin repot instalasi lokal, **Google Colab** adalah pilihan sangat praktis. Di outline naskah utama, Colab sudah disebut sebagai salah satu editor/lingkungan belajar yang patut dipertimbangkan sejak awal.

Langkah memakai Google Colab

1. Buka browser
2. Masuk ke akun Google
3. Buka Google Colab
4. Buat notebook baru
5. Jalankan kode di *cell* seperti di Jupyter

Kelebihan Colab untuk pemula

- tidak perlu instal Python di laptop
- tidak perlu setup `pip` dan `venv` di awal
- bisa langsung menulis dan menjalankan kode
- cocok untuk latihan cepat, tugas, dan eksperimen awal

Kekurangannya

- bergantung pada koneksi internet
- struktur proyek lokal tidak terasa sekuat Ubuntu
- kebiasaan kerja profesional seperti `venv` dan manajemen file lokal kurang terlatih

Contoh kode pertama di Colab

```
print("Halo dari Google Colab!")
```

Contoh data sederhana:

```
import pandas as pd

df = pd.DataFrame({
    "nama": ["Andi", "Budi", "Citra"],
    "nilai": [88, 75, 93]
})

df
```

Contoh visualisasi:

```
import matplotlib.pyplot as plt

df.plot(x="nama", y="nilai", kind="bar")
plt.title("Nilai Mahasiswa")
plt.show()
```

Dengan Colab, pembaca bisa langsung masuk ke praktik tanpa hambatan teknis. Setelah nyaman, barulah pembaca disarankan pindah ke Ubuntu 26.04 agar terbiasa dengan lingkungan kerja yang lebih mandiri.

2.8. Langkah cepat dari nol sampai siap praktik

Agar lebih mudah diikuti, berikut ringkasan langkah yang paling praktis.

Opsi Ubuntu 26.04

- instal `python3`, `python3-pip`, `python3-venv`
- buat folder proyek
- buat dan aktifkan `.venv`
- instal `numpy`, `pandas`, `matplotlib`, `scikit-learn`, `notebook`
- buat struktur folder proyek
- jalankan file `.py`
- jalankan notebook `.ipynb`

Opsi Google Colab

- buka browser
- masuk ke Colab
- buat notebook baru
- coba kode Python pertama
- lanjut ke latihan dan eksperimen

Dengan dua jalur ini, pembaca punya kebebasan untuk menyesuaikan dengan kondisi perangkat dan kenyamanan belajar.

2.9. Tips menghindari error pemula

Outline utama sudah menekankan beberapa error umum: masalah PATH, *package* tidak ditemukan, dan kernel salah. Itu memang sangat sering terjadi di tahap awal.

Beberapa tips penting:

- **Kalau `import pandas` gagal**, kemungkinan `pandas` belum terpasang atau `venv` belum aktif
- **Kalau notebook tidak menemukan *library***, kemungkinan kernel salah
- **Kalau VS Code berbeda perilaku dengan terminal**, kemungkinan interpreter salah
- **Kalau file data tidak terbaca**, cek lokasi folder kerja
- **Biasakan melihat `(.venv)` di terminal** sebelum memasang *library*

Tips ini sederhana, tetapi menyelamatkan banyak waktu dan frustrasi.

Penutup Bab

Bab ini menegaskan satu prinsip penting: **setup cukup seperlunya, lalu segera praktik**. Untuk pembaca buku ini, ada dua jalur yang sama-sama valid. **Ubuntu 26.04** cocok bagi yang ingin fondasi lebih kuat, rapi, dan profesional. **Google Colab** cocok bagi yang ingin cepat mulai tanpa instalasi. Keduanya mendukung tujuan besar buku ini, yaitu membawa pembaca dari nol menuju kemampuan bekerja dengan Python, data, statistik, dan *machine learning* secara bertahap. Outline naskah utama sendiri jelas mengarahkan bahwa Bab 2 harus berfungsi sebagai pintu masuk yang nyaman, bukan hambatan teknis yang melelahkan.

Tantangan kecil untuk pembaca: pilih salah satu jalur hari ini. Bila memakai Ubuntu, buat proyek pertama dengan `.venv` dan jalankan satu file `.py`. Bila memakai Colab, buat satu notebook dan tampilkan tabel kecil berisi tiga nama dan tiga nilai. Begitu langkah kecil ini berhasil, Anda sudah resmi masuk ke dunia Python dengan cara yang sehat, praktis, dan menyenangkan.

Bab 3. Python Dasar untuk Calon Praktisi AI

Bab ini adalah **fondasi kerja** untuk seluruh isi buku. Sebelum pembaca masuk ke *data analysis*, statistik, *machine learning*, atau AI yang lebih lanjut, ada satu hal yang harus benar-benar dikuasai lebih dulu: **cara berpikir dengan Python**. Di outline naskah utama, bagian fondasi Python memang dipecah secara bertahap menjadi variabel, tipe data, kontrol alur, struktur data, fungsi, file handling, dan debugging. Arah ini sangat tepat, karena calon praktisi AI tidak cukup hanya "bisa menjalankan notebook", tetapi perlu memahami apa yang sedang dilakukan oleh kode.

Materi bab ini juga selaras dengan referensi lampiran yang menempatkan **syntax, variables, data types, control flow, functions, data structures, file handling, dan exception handling** sebagai inti awal pembelajaran Python. Buku Abdullah secara eksplisit merangkum Python Basics, Data Structures, Control Flow, Functions, Exception Handling, dan File Handling sebagai pilar utama sebelum pembaca bergerak ke data dan AI. Buku Jain juga menegaskan bahwa untuk masuk ke analisis data, pembaca harus lebih dulu menguasai **syntax, variables, data types, control flow, functions, dan modules**.

Bab ini sengaja ditulis dengan gaya **mudah, praktis, dan membumi**. Tujuannya bukan membuat pembaca hafal semua teori, melainkan agar pembaca bisa:

- **menyimpan data ke dalam variabel**
- **mengolah logika dengan operator**
- **membuat keputusan dengan `if`**
- **mengulang pekerjaan dengan `for` dan `while`**
- **menyimpan data dalam `list`, `tuple`, `set`, dan `dict`**
- **membuat fungsi sederhana**
- **membaca file teks, CSV, dan JSON**
- **memahami error dasar dan melakukan debugging sederhana**

Kalau pembaca menguasai bab ini dengan baik, maka bab-bab berikutnya tentang NumPy, Pandas, statistik, dan ML akan terasa jauh lebih mudah.

3.1 Mengapa Python dasar penting untuk calon praktisi AI

Banyak pemula tergoda untuk langsung belajar model AI, *prompt engineering*, atau *deep learning*, tetapi sering tersendat karena fondasi Python-nya belum kuat. Mereka mungkin bisa menyalin kode dari internet, tetapi tidak paham mengapa ada variabel tertentu, mengapa data harus berbentuk `dict`, mengapa perlu fungsi, atau kenapa program berhenti karena *error*. Karena itu, naskah utama menempatkan Python dasar sebagai **fondasi untuk analisis data**, bukan sekadar bab formalitas. Setelah memahami Python dasar, pembaca akan lebih siap membaca dataset, membersihkan data, membuat visualisasi, dan melatih model.

Dalam praktik AI/ML, hampir semua pekerjaan berawal dari struktur sederhana. Misalnya:

- data penjualan disimpan dalam **variabel**
- kumpulan nilai disimpan dalam **list**
- data per orang sering disimpan dalam **dictionary**
- proses pembersihan data memakai **loop**
- validasi input memakai **if**
- langkah berulang dikemas dalam **fungsi**
- dataset sering dibaca dari **CSV** atau **JSON**
- program yang kuat membutuhkan **error handling**

Artinya, Python dasar bukan pengetahuan terpisah dari AI. Justru Python dasar adalah **jalan masuk utama** untuk bekerja dengan data dan model.

3.2 Variabel, tipe data, dan operator

Apa itu variabel

Variabel adalah **wadah untuk menyimpan nilai**. Dalam Python, variabel bisa berisi angka, teks, nilai benar/salah, daftar data, atau struktur lain. Buku Jain menekankan bahwa Python dikenal ramah pemula karena *syntax*-nya sederhana dan mudah dibaca, termasuk dalam hal mendefinisikan variabel.

Contoh paling sederhana:

```
nama = "Andi"
umur = 20
ipk = 3.75
aktif = True
```

Di sini:

- `nama` berisi teks atau `str`
- `umur` berisi bilangan bulat atau `int`
- `ipk` berisi bilangan pecahan atau `float`
- `aktif` berisi nilai logika atau `bool`

Mengapa variabel penting

Dalam dunia AI dan data, variabel dipakai untuk menyimpan hampir semua hal:

- nama kolom
- jumlah data
- rata-rata nilai
- hasil prediksi
- status proses
- parameter model

Tanpa variabel, program akan menjadi kaku dan sulit dibaca.

Contoh di Ubuntu 26.04

Buat file `src/variabel_dasar.py`:

```
nama = "Citra"
umur = 21
tinggi = 162.5
siap_belajar_ai = True

print("Nama:", nama)
print("Umur:", umur)
print("Tinggi:", tinggi)
print("Siap belajar AI:", siap_belajar_ai)
```

Jalankan:

```
python src/variabel_dasar.py
```

Contoh keluaran:

```
Nama: Citra
Umur: 21
Tinggi: 162.5
Siap belajar AI: True
```

Tipe data dasar

Empat tipe data paling penting di tahap awal adalah:

- **`int`** untuk bilangan bulat
- **`float`** untuk bilangan pecahan
- **`str`** untuk teks
- **`bool`** untuk nilai `True` atau `False`

Contoh:

```
jumlah_mahasiswa = 120
akurasi_model = 0.91
judul = "Pengantar Python"
lulus = True
```

Mengecek tipe data

Python menyediakan `type()` untuk melihat tipe data.

```
print(type(jumlah_mahasiswa))
print(type(akurasi_model))
print(type(judul))
print(type(lulus))
```

Operator dasar

Operator dipakai untuk melakukan perhitungan dan logika. Buku Abdullah merangkum operators sebagai bagian penting Python Basics, dan referensi lain juga memakai operasi dasar ini untuk contoh-contoh analisis praktis.

Operator aritmetika

- `+` tambah
- `-` kurang
- `*` kali
- `/` bagi
- `//` pembagian bulat
- `%` sisa bagi
- `**` pangkat

Contoh:

```
a = 10
b = 3

print(a + b)   # 13
print(a - b)   # 7
print(a * b)   # 30
print(a / b)   # 3.333...
print(a // b)  # 3
print(a % b)   # 1
print(a ** b)  # 1000
```

Operator perbandingan

- `==` sama dengan
- `!=` tidak sama dengan
- `>` lebih besar
- `<` lebih kecil
- `>=` lebih besar atau sama
- `<=` lebih kecil atau sama

Contoh:

```
nilai = 85
print(nilai > 75)   # True
print(nilai == 100) # False
```

Operator logika

- `and`
- `or`
- `not`

Contoh:

```
nilai = 85
kehadiran = 90

print(nilai >= 75 and kehadiran >= 80)
print(nilai >= 90 or kehadiran >= 90)
print(not nilai < 75)
```

Contoh praktis: menghitung nilai akhir

```
tugas = 80
uts = 78
uas = 90

nilai_akhir = (0.3 * tugas) + (0.3 * uts) + (0.4 *
uas)
print("Nilai akhir:", nilai_akhir)
```

Ini adalah contoh kecil, tetapi sangat penting. Banyak analisis data sesungguhnya berawal dari logika seperti ini: menyimpan variabel, menghitung, lalu menampilkan hasil.

3.3 if, for, dan while

Setelah memahami variabel dan operator, langkah berikutnya adalah mengatur **alur program**. Inilah inti dari *control flow*. Referensi Abdullah dan Jain sama-sama menempatkan *conditional statements* serta *looping* sebagai bagian inti Python dasar.

if: membuat keputusan

`if` dipakai ketika program harus memilih tindakan berdasarkan kondisi.

Contoh sederhana:

```
nilai = 82

if nilai >= 75:
    print("Lulus")
else:
    print("Belum lulus")
```

if, elif, else

Kalau kondisi lebih dari satu:

```
nilai = 88

if nilai >= 90:
    print("Predikat A")
elif nilai >= 80:
    print("Predikat B")
elif nilai >= 70:
    print("Predikat C")
else:
    print("Perlu perbaikan")
```

Contoh praktis: validasi prediksi sederhana

```
akurasi = 0.87

if akurasi >= 0.9:
    print("Model sangat baik")
elif akurasi >= 0.8:
    print("Model cukup baik")
else:
    print("Model perlu diperbaiki")
```

for: mengulang data

`for` dipakai saat kita ingin memproses setiap item dalam kumpulan data.

```
angka = [10, 20, 30, 40]

for item in angka:
    print("Nilai:", item)
```

Contoh praktis: menjumlah nilai

```
nilai = [80, 75, 90, 88]
total = 0

for n in nilai:
    total = total + n

print("Total nilai:", total)
print("Rata-rata:", total / len(nilai))
```

while: mengulang selama kondisi masih benar

`while` cocok untuk proses yang belum tahu pasti berapa kali akan berjalan.

```
i = 1

while i <= 5:
    print("Perulangan ke-", i)
    i += 1
```

Kapan memakai for dan kapan while

Secara praktis:

- gunakan **`for`** saat mengolah data yang jumlah elemennya sudah jelas
- gunakan **`while`** saat pengulangan bergantung pada kondisi

Contoh mini studi kasus

Misalnya kita ingin memeriksa daftar nilai dan menampilkan siapa yang lulus.

```
data_nilai = {
    "Andi": 78,
    "Budi": 65,
    "Citra": 92,
    "Dina": 74
}

for nama, nilai in data_nilai.items():
    if nilai >= 75:
        print(nama, "lulus")
    else:
        print(nama, "belum lulus")
```

Kode seperti ini nanti sangat berguna ketika pembaca mulai memproses dataset atau hasil model.

3.4 `list`, `tuple`, `set`, dan `dict`

Struktur data adalah cara Python menyimpan kumpulan nilai. Buku Abdullah menandai **lists, tuples, sets, and dictionaries** sebagai inti Data Structures, dan naskah utama juga memang memisahkan struktur data sebagai materi penting sebelum masuk ke dunia data nyata.

list

`list` adalah struktur data paling umum. Ia bisa diubah, ditambah, dihapus, dan diakses dengan indeks.

```
buah = ["apel", "jeruk", "mangga"]
print(buah[0])
```

Tambahkan data:

```
buah.append("pisang")
print(buah)
```

Contoh praktis:

```
nilai = [88, 75, 93, 81]
print("Jumlah data:", len(nilai))
print("Nilai tertinggi:", max(nilai))
print("Nilai terendah:", min(nilai))
```

tuple

`tuple` mirip `list`, tetapi **tidak bisa diubah** setelah dibuat.

```
koordinat = (10, 20)
print(koordinat[0])
```

Tuple sering dipakai untuk data yang sifatnya tetap, misalnya pasangan `(x, y)` atau konfigurasi tertentu.

set

`set` dipakai untuk kumpulan nilai unik. Ia berguna saat kita ingin menghapus duplikasi.

```
angka = {1, 2, 2, 3, 4, 4, 5}
print(angka)
```

Contoh hasil:

```
{1, 2, 3, 4, 5}
```

dict

`dict` atau dictionary menyimpan data dalam bentuk pasangan **kunci: nilai**. Ini sangat penting untuk dunia data dan AI, karena banyak data nyata berbentuk seperti ini.

```
mahasiswa = {
    "nama": "Andi",
    "umur": 20,
    "jurusan": "Informatika"
}

print(mahasiswa["nama"])
```

Tambahkan data baru:

```
mahasiswa["ipk"] = 3.72
print(mahasiswa)
```

Mengapa dict sangat penting

Dalam praktik AI dan data, dictionary berguna untuk:

- menyimpan data per entitas
- menyimpan parameter
- mengelompokkan hasil
- memetakan label ke nilai

Contoh praktis gabungan

```
data_siswa = [
    {"nama": "Andi", "nilai": 80},
    {"nama": "Budi", "nilai": 70},
    {"nama": "Citra", "nilai": 95}
]

for siswa in data_siswa:
    if siswa["nilai"] >= 75:
        print(siswa["nama"], "lulus")
    else:
        print(siswa["nama"], "belum lulus")
```

Inilah salah satu pola paling penting dalam Python: **list berisi dictionary**. Pola ini nanti sangat sering muncul saat membaca data JSON atau bekerja dengan record data nyata.

3.5 Fungsi

Setelah pembaca mulai menulis kode yang lebih panjang, akan muncul kebutuhan untuk **merapikan logika**. Di sinilah fungsi menjadi sangat penting. Referensi Jain menekankan bahwa fungsi yang baik sebaiknya kecil, fokus pada satu tugas, dan diberi *docstring* agar tujuan serta cara pakainya jelas. Referensi lain juga menyebut fungsi sebagai alat untuk membuat kode lebih profesional, mudah dirawat, dan mudah dipakai ulang.

Apa itu fungsi

Fungsi adalah blok kode yang bisa dipanggil berulang kali.

Contoh dasar:

```
def sapa():
    print("Halo, selamat belajar Python!")
```

Panggil fungsi:

```
sapa()
```

Fungsi dengan parameter

```
def sapa_nama(nama):
    print("Halo,", nama)

sapa_nama("Dina")
```

Fungsi dengan return

Kalau fungsi perlu menghasilkan nilai, gunakan `return`.

```
def hitung_rata_rata(a, b, c):
    return (a + b + c) / 3

hasil = hitung_rata_rata(80, 90, 70)
print("Rata-rata:", hasil)
```

Contoh fungsi untuk calon praktisi AI

```
def klasifikasi_kelulusan(nilai):
    if nilai >= 75:
        return "Lulus"
    return "Belum lulus"

print(klasifikasi_kelulusan(88))
print(klasifikasi_kelulusan(60))
```

Fungsi kecil lebih baik daripada fungsi besar

Praktik baik yang disarankan:

- satu fungsi untuk satu tugas
- nama fungsi jelas
- parameter secukupnya
- hasil dikembalikan dengan `return`

Contoh fungsi yang lebih realistis

```
def hitung_statistik_sederhana(data):
    total = sum(data)
    jumlah = len(data)
    rata_rata = total / jumlah
    return total, jumlah, rata_rata

nilai = [80, 85, 90, 75, 88]
total, jumlah, rata = hitung_statistik_sederhana(nilai)

print("Total:", total)
print("Jumlah data:", jumlah)
print("Rata-rata:", rata)
```

Fungsi seperti ini nanti akan sangat berguna saat pembaca mulai menulis analisis yang lebih panjang.

3.6 Membaca file teks, CSV, dan JSON

Di dunia nyata, data jarang ditulis satu per satu di dalam kode. Data biasanya berasal dari file. Karena itu, *file handling* menjadi salah satu keterampilan inti. Buku Abdullah menyebut **Reading and writing files** sebagai bagian penting fondasi Python, dan referensi lain menjelaskan bahwa CSV dan JSON adalah format yang sangat umum untuk berbagi data. Saltz secara khusus menekankan JSON sebagai format yang terstruktur dan mudah dibaca manusia, serta lazim digunakan di web dan API.

A. Membaca file teks

Misalnya kita punya file `data/raw/catatan.txt` berisi:

```
Python itu menyenangkan.
Belajar data butuh latihan.
AI dimulai dari fondasi yang kuat.
```

Buat script:

```
with open("data/raw/catatan.txt", "r", encoding="utf-8")
as file:
    isi = file.read()

print(isi)
```

Membaca baris per baris

```
with open("data/raw/catatan.txt", "r", encoding="utf-8") as file:
    for baris in file:
        print(baris.strip())
```

Menulis file teks

```
teks = "Ini catatan belajar Python di Ubuntu 26.04."

with open("outputs/hasil_catatan.txt", "w", encoding="utf-8") as file:
    file.write(teks)
```

B. Membaca CSV

CSV sangat penting karena banyak dataset awal disimpan dalam format ini. Referensi Van Der Post mencontohkan workflow membaca CSV, membersihkan data, menghitung metrik, dan menulis hasil kembali.

Buat file `data/raw/nilai.csv`:

```
nama,nilai
Andi,88
Budi,75
Citra,93
```

Baca dengan modul bawaan `csv`:

```
import csv

with open("data/raw/nilai.csv", "r", encoding="utf-8") as file:
    reader = csv.DictReader(file)
    for row in reader:
        print(row["nama"], row["nilai"])
```

Namun untuk dunia data, pembaca juga perlu tahu bahwa CSV sangat sering dibaca dengan `pandas`.

```
import pandas as pd

df = pd.read_csv("data/raw/nilai.csv")
print(df)
print("Rata-rata nilai:", df["nilai"].mean())
```

C. Membaca JSON

JSON banyak dipakai untuk konfigurasi, pertukaran data, dan API. Saltz menegaskan bahwa JSON adalah format yang terstruktur dan mudah dibaca manusia, serta sangat umum di web.

Buat file `data/raw/mahasiswa.json`:

```
[
    {"nama": "Andi", "nilai": 88},
    {"nama": "Budi", "nilai": 75},
    {"nama": "Citra", "nilai": 93}
]
```

Baca dengan Python:

```
import json

with open("data/raw/mahasiswa.json", "r", encoding="utf-8") as
file:
    data = json.load(file)

for item in data:
    print(item["nama"], item["nilai"])
```

Menulis JSON

```
import json

hasil = {
    "model": "Linear Regression",
    "akurasi": 0.91,
    "status": "baik"
}

with open("outputs/hasil_model.json", "w",
encoding="utf-8") as file:
    json.dump(hasil, file, indent=4)
```

Kemampuan membaca file teks, CSV, dan JSON ini akan menjadi bekal yang sangat penting untuk bab-bab berikutnya.

3.7 Error dasar dan debugging sederhana

Setiap orang yang belajar Python akan bertemu error. Itu normal. Referensi Van Der Post menekankan bahwa error handling dan debugging adalah komponen esensial dari proses coding, dan error bisa dibagi menjadi **syntax errors, runtime errors, dan logical errors**. Referensi lain juga menekankan penggunaan `try-except` serta *print debugging* sebagai pendekatan awal yang sangat efektif.

A. Jenis error dasar

1. *Syntax error*

Terjadi ketika penulisan kode salah.

```
print("Halo"
```

Ini akan gagal karena tanda kurung penutup kurang.

2. *Runtime error*

Terjadi saat program dijalankan.

```
angka = 10 / 0
```

Ini memicu `ZeroDivisionError`.

3. *Logical error*

Program jalan, tetapi hasilnya salah.

```
nilai = [80, 90, 100]
rata = sum(nilai) / 2
print(rata)
```

Kode ini tidak error, tetapi logikanya salah karena seharusnya dibagi 3, bukan 2.

B. Memakai try-except

Python menyediakan mekanisme untuk menangani error dengan lebih anggun. Referensi AI with Python menunjukkan `try-except`, penanganan banyak exception, serta `else` dan `finally` sebagai pola yang baik untuk membuat program lebih tangguh.

Contoh dasar:

```
try:
    hasil = 10 / 0
except ZeroDivisionError:
```

```
    print("Error: pembagian dengan nol tidak
diperbolehkan")
```

Menangani beberapa jenis error

```
try:
    nilai = int("abc")
except (ValueError, TypeError):
    print("Input tidak valid")
```

`else` dan `finally`

```
try:
    angka = int("123")
except ValueError:
    print("Konversi gagal")
else:
    print("Konversi berhasil:", angka)
finally:
    print("Program selesai")
```

C. Debugging dengan print()

Salah satu teknik termudah adalah menambahkan `print()` di titik-titik penting. Referensi juga menekankan bahwa *print statements* sering menjadi langkah pertama debugging sebelum memakai alat yang lebih canggih.

Contoh:

```
def cari_maksimum(data):
    max_val = data[0]
    for item in data:
        print("Item saat ini:", item, "| Maks saat ini:", max_val)
        if item > max_val:
            max_val = item
    return max_val

angka = [2, 4, 6, 1, 9]
print("Maksimum:", cari_maksimum(angka))
```

Dengan cara ini, pembaca bisa melihat alur logika program.

D. Pola debugging sederhana yang baik

Mengacu pada pola umum yang juga dijelaskan di referensi, debugging bisa dilakukan dengan langkah berikut:

- **Identifikasi** masalah
  Apakah ada pesan error, hasil aneh, atau output kosong?
- **Isolasi** sumber masalah
  Baris mana yang memicu error? Variabel mana yang nilainya tidak sesuai?
- **Perbaiki**
  Ubah kode, tipe data, path file, atau logika kondisi.
- **Uji ulang**
  Jalankan lagi dan pastikan perbaikannya benar.

E. Contoh mini debugging

```
nilai = [80, 90, 100]
total = 0

for n in nilai:
    total += n

rata_rata = total / len(nilai)
print("Rata-rata:", rata_rata)
```

Kalau sebelumnya hasil salah, pembaca bisa mengecek:

- apakah `nilai` benar
- apakah `total` bertambah dengan benar
- apakah pembagi menggunakan `len(nilai)`

Inilah inti debugging: **jangan panik, tetapi telusuri logika secara sistematis**.

3.8 Langkah praktik yang disarankan untuk pembaca

Agar bab ini benar-benar hidup, berikut langkah praktik yang sangat disarankan.

Tahap 1 — Variabel dan operator

- buat file `src/latihan_variabel.py`
- simpan data nama, umur, nilai, dan status
- coba operasi aritmetika dan logika
- tampilkan hasil dengan `print()`

Tahap 2 — if, for, while

- buat keputusan lulus/tidak lulus
- ulangi daftar nilai dengan `for`
- buat hitungan sederhana dengan `while`

Tahap 3 — Struktur data

- simpan data dalam `list`
- ubah menjadi `dict`
- buat `list` yang berisi beberapa `dict`

Tahap 4 — Fungsi

- buat fungsi hitung rata-rata
- buat fungsi cek kelulusan
- panggil fungsi dengan data yang berbeda

Tahap 5 — File handling

- baca file teks
- baca file CSV
- baca file JSON
- simpan hasil sederhana ke file keluaran

Tahap 6 — Debugging

- sengaja buat satu error kecil
- baca pesan error
- perbaiki
- tambah `print()` untuk menelusuri logika

Urutan seperti ini akan membuat pembaca tidak hanya memahami teori, tetapi juga benar-benar membangun kebiasaan kerja.

3.9 Mini proyek kecil: data nilai mahasiswa

Sebagai penutup bab, pembaca bisa mencoba mini proyek kecil berikut.

Buat file `src/mini_project_nilai.py`:

```
import csv

def tentukan_status(nilai):
    if nilai >= 75:
        return "Lulus"
    return "Belum lulus"

data_mahasiswa = []

with open("data/raw/nilai.csv", "r", encoding="utf-8") as file:
    reader = csv.DictReader(file)
    for row in reader:
        nama = row["nama"]
        nilai = int(row["nilai"])
        status = tentukan_status(nilai)

        data_mahasiswa.append({
            "nama": nama,
            "nilai": nilai,
            "status": status
        })

for mhs in data_mahasiswa:
    print(f'{mhs["nama"]} | {mhs["nilai"]} | {mhs["status"]}')
```

Apa yang dilatih dari proyek kecil ini?

- variabel
- tipe data
- `if`
- `for`
- `dict`
- fungsi
- membaca CSV

Inilah contoh yang sangat baik tentang bagaimana Python dasar mulai terasa relevan untuk data dan AI.

Penutup Bab

Bab ini adalah **pondasi mental dan teknis** bagi calon praktisi AI. Pembaca tidak hanya diperkenalkan pada variabel, tipe data, operator, *control flow*, struktur data, fungsi, file handling, dan debugging, tetapi juga diajak melihat bagaimana semua elemen itu terhubung dalam pekerjaan nyata. Naskah utama memang secara sengaja membangun jalur belajar dari Python dasar menuju analisis data, statistik, dan *machine learning*, dan bab ini adalah batu pijakan pertama yang harus kokoh.

Pesan paling penting dari bab ini sederhana: **jangan buru-buru mengejar AI yang rumit jika Python dasar belum nyaman di tangan**. Ketika pembaca sudah akrab dengan variabel, struktur data, fungsi, file, dan error, maka perjalanan ke data science dan AI akan terasa jauh lebih masuk akal, lebih menyenangkan, dan lebih percaya diri.

Tantangan kecil untuk pembaca: buat satu program di Ubuntu 26.04 yang membaca file CSV nilai mahasiswa, menghitung rata-rata, lalu menampilkan siapa yang lulus dan siapa yang belum. Kalau Anda bisa menyelesaikannya sendiri, berarti fondasi Python Anda sudah mulai terbentuk dengan baik.

Bab 4. Mini Project 1 — Python Dasar untuk Mengolah Data Sederhana

Bab ini adalah **jembatan pertama** dari belajar sintaks Python menuju **berpikir berbasis data**. Di outline naskah utama, mini project pertama memang dirancang sebagai **analisis penjualan toko kecil tanpa *library* berat**, dengan fokus pada **input data sederhana, perhitungan total, rata-rata, produk terlaris, ekspor hasil ke CSV/JSON, dan refleksi tentang bagaimana coding dasar berubah menjadi cara berpikir data**. Penempatan mini project di tahap awal ini sangat tepat, karena pembaca tidak hanya belajar variable, `if`, `for`, `list`, `dict`, fungsi, dan file handling secara terpisah, tetapi mulai merasakan bagaimana semuanya dipakai bersama untuk menyelesaikan masalah nyata.

Arah seperti ini juga sejalan dengan materi referensi lain yang menekankan bahwa fondasi Python untuk analisis data harus dibangun lewat **script sederhana, pengolahan file CSV, perhitungan metrik, penulisan hasil, dan dokumentasi langkah kerja**. Salah satu materi lampiran bahkan memberi pola latihan yang sangat mirip: **membaca data CSV, membersihkan atau mengolah data, menghitung metrik kunci, lalu menulis hasil kembali ke file**. Pendekatan ini sangat cocok untuk pemula karena hasilnya terlihat, logikanya jelas, dan prosesnya terasa relevan dengan dunia nyata.

Mini project ini sengaja dibuat **tanpa NumPy, tanpa Pandas, dan tanpa scikit-learn**. Tujuannya adalah melatih pembaca memakai **Python murni** lebih dulu. Dengan demikian, saat nanti pindah ke Pandas atau notebook yang lebih canggih, pembaca sudah paham logikanya, bukan sekadar menyalin perintah. Inilah tahap ketika pembaca mulai menyadari bahwa Python bukan hanya alat untuk “menjalankan AI”, tetapi juga alat untuk **membaca masalah, menghitung, merangkum, dan mengambil kesimpulan dari data**.

Tujuan mini project

Setelah menyelesaikan mini project ini, pembaca diharapkan mampu:

- **menyimpan data penjualan sederhana** ke dalam struktur Python
- **menghitung total penjualan**
- **menghitung rata-rata penjualan**
- **menemukan nilai maksimum dan minimum**
- **menentukan produk terlaris dan produk dengan penjualan terendah**
- **menyimpan hasil ringkasan ke file CSV**
- **mulai berpikir seperti analis data**, meskipun masih memakai Python dasar

Mini project ini juga membantu menguatkan banyak materi dari bab sebelumnya, terutama:

- variabel dan tipe data
- operator aritmetika
- `if`, `for`
- `list` dan `dict`
- fungsi
- membaca dan menulis file

4.1 Skenario kasus: toko kecil dengan data penjualan sederhana

Bayangkan ada sebuah toko kecil yang menjual beberapa produk harian. Pemilik toko belum punya sistem analitik canggih. Data penjualan masih sangat sederhana, misalnya hanya berupa nama produk, jumlah terjual, dan harga satuan. Dari data sekecil itu, sebenarnya kita sudah bisa menghitung banyak hal penting:

- total omzet
- rata-rata omzet per produk
- produk paling laku
- produk paling kecil penjualannya
- ringkasan yang bisa disimpan untuk laporan

Di sinilah inti berpikir data dimulai. **Data yang sederhana tetap bisa menghasilkan insight**, asalkan kita tahu cara mengolahnya.

Sebagai contoh, kita akan memakai data berikut:

Produk	Jumlah Terjual	Harga Satuan
Beras 5kg	12	74000
Gula 1kg	20	17000
Minyak 1L	15	21000
Telur 1kg	18	28000
Mi Instan	40	3500

Dari tabel kecil ini, kita akan menghitung:

- omzet tiap produk
- total omzet semua produk
- rata-rata omzet
- omzet maksimum
- omzet minimum
- produk terlaris berdasarkan jumlah unit
- produk dengan omzet tertinggi
- hasil ringkasan ke CSV

4.2 Konsep logika sebelum coding

Sebelum menulis kode, pembaca perlu belajar satu kebiasaan yang sangat penting: **pikirkan langkah logikanya dulu**. Ini adalah inti dari pemrograman yang sehat.

Untuk kasus mini project ini, urutannya sederhana:

1. Siapkan data penjualan.
2. Hitung omzet setiap produk dengan rumus:
   omzet = jumlah_terjual × harga_satuan
3. Simpan hasil omzet ke data masing-masing produk.
4. Hitung total seluruh omzet.
5. Hitung rata-rata omzet.
6. Cari nilai omzet tertinggi dan terendah.
7. Cari produk dengan jumlah terjual paling banyak.
8. Simpan hasil ke file CSV.
9. Tulis refleksi kecil: apa yang kita pelajari dari data ini?

Urutan seperti ini sangat penting karena dalam dunia data dan AI, pekerjaan hampir selalu dimulai dari hal yang sama: **pahami masalah → tentukan data → proses → hasilkan ringkasan → interpretasi**. Outline naskah utama juga menekankan pentingnya *workflow* seperti ini agar pembaca tidak sekadar menjalankan kode, tetapi memahami urutan kerja yang benar.

4.3 Struktur data yang dipakai

Untuk mini project ini, struktur data yang paling cocok adalah **list berisi dictionary**. Ini adalah pola Python yang sangat penting untuk calon praktisi data.

Setiap produk akan disimpan sebagai satu dictionary, misalnya:

```
{
    "produk": "Beras 5kg",
    "jumlah": 12,
    "harga": 74000
}
```

Lalu semua produk dikumpulkan dalam satu list.

Kenapa bentuk ini bagus?

- mudah dibaca manusia
- mirip struktur data JSON
- cocok untuk diiterasi dengan `for`
- mudah ditambah kolom baru seperti `omzet`
- nanti mudah dihubungkan ke CSV atau Pandas

Pola seperti ini juga sangat selaras dengan pembahasan Python dasar di outline utama, khususnya **list**, **dict**, dan **nested structure sederhana** yang dipakai untuk menyimpan data siswa, transaksi, atau produk.

4.4 Source code Python lengkap

Berikut adalah **source code Python lengkap** untuk mini project ini. Kode ini bisa dijalankan di Ubuntu 26.04 dengan Python 3 biasa, tanpa *library* tambahan.

Simpan sebagai file:

```
src/mini_project_penjualan.py
```

Lalu isi dengan kode berikut:

```
import csv

# ========================================
# MINI PROJECT 1
# Python Dasar untuk Mengolah Data Sederhana
# Studi kasus: data penjualan toko kecil
# ========================================

# 1. Data awal
data_penjualan = [
    {"produk": "Beras 5kg", "jumlah": 12, "harga": 74000},
    {"produk": "Gula 1kg", "jumlah": 20, "harga": 17000},
    {"produk": "Minyak 1L", "jumlah": 15, "harga": 21000},
    {"produk": "Telur 1kg", "jumlah": 18, "harga": 28000},
    {"produk": "Mi Instan", "jumlah": 40, "harga": 3500},
]

# 2. Fungsi untuk menghitung omzet per produk
def hitung_omzet_produk(jumlah, harga):
    return jumlah * harga


# 3. Fungsi untuk menghitung total omzet
def hitung_total_omzet(data):
    total = 0
    for item in data:
        total += item["omzet"]
    return total


# 4. Fungsi untuk menghitung rata-rata omzet
def hitung_rata_rata_omzet(data):
    if len(data) == 0:
        return 0
    return hitung_total_omzet(data) / len(data)


# 5. Fungsi mencari produk dengan omzet tertinggi
def cari_produk_omzet_tertinggi(data):
    produk_tertinggi = data[0]
    for item in data:
```

```
        if item["omzet"] > produk_tertinggi["omzet"]:
            produk_tertinggi = item
    return produk_tertinggi


# 6. Fungsi mencari produk dengan omzet terendah
def cari_produk_omzet_terendah(data):
    produk_terendah = data[0]
    for item in data:
        if item["omzet"] < produk_terendah["omzet"]:
            produk_terendah = item
    return produk_terendah


# 7. Fungsi mencari produk terlaris berdasarkan jumlah terjual
def cari_produk_terlaris(data):
    produk_terlaris = data[0]
    for item in data:
        if item["jumlah"] > produk_terlaris["jumlah"]:
            produk_terlaris = item
    return produk_terlaris


# 8. Tambahkan kolom omzet ke setiap produk
for item in data_penjualan:
    item["omzet"] = hitung_omzet_produk(item["jumlah"], item["harga"])


# 9. Hitung ringkasan
total_omzet = hitung_total_omzet(data_penjualan)
rata_rata_omzet = hitung_rata_rata_omzet(data_penjualan)
produk_omzet_tertinggi = cari_produk_omzet_tertinggi(data_penjualan)
produk_omzet_terendah = cari_produk_omzet_terendah(data_penjualan)
produk_terlaris = cari_produk_terlaris(data_penjualan)

nilai_maksimum = produk_omzet_tertinggi["omzet"]
nilai_minimum = produk_omzet_terendah["omzet"]


# 10. Tampilkan hasil ke layar
print("=" * 60)
print("RINGKASAN DATA PENJUALAN TOKO KECIL")
print("=" * 60)

for item in data_penjualan:
    print(
        f'Produk: {item["produk"]:<12} | '
        f'Jumlah: {item["jumlah"]:>3} | '
        f'Harga: Rp{item["harga"]:>7,} | '
        f'Omzet: Rp{item["omzet"]:>8,}'
    )

print("-" * 60)
print(f"Total omzet        : Rp{total_omzet:,.0f}")
print(f"Rata-rata omzet    : Rp{rata_rata_omzet:,.2f}")
print(f"Nilai maksimum     : Rp{nilai_maksimum:,.0f}")
print(f"Nilai minimum      : Rp{nilai_minimum:,.0f}")
print(f'Produk terlaris    : {produk_terlaris["produk"]}
({produk_terlaris["jumlah"]} unit)')
```

```
print(
    f'Omzet tertinggi    : {produk_omzet_tertinggi["produk"]} '
    f'(Rp{produk_omzet_tertinggi["omzet"]:,.0f})'
)
print(
    f'Omzet terendah     : {produk_omzet_terendah["produk"]} '
    f'(Rp{produk_omzet_terendah["omzet"]:,.0f})'
)
print("=" * 60)


# 11. Simpan data detail ke CSV
with open("outputs/penjualan_detail.csv", "w", newline="", encoding="utf-8")
as file_csv:
    fieldnames = ["produk", "jumlah", "harga", "omzet"]
    writer = csv.DictWriter(file_csv, fieldnames=fieldnames)

    writer.writeheader()
    for item in data_penjualan:
        writer.writerow(item)


# 12. Simpan ringkasan ke CSV
with open("outputs/ringkasan_penjualan.csv", "w", newline="",
encoding="utf-8") as file_ringkasan:
    writer = csv.writer(file_ringkasan)
    writer.writerow(["metrik", "nilai"])
    writer.writerow(["total_omzet", total_omzet])
    writer.writerow(["rata_rata_omzet", rata_rata_omzet])
    writer.writerow(["nilai_maksimum", nilai_maksimum])
    writer.writerow(["nilai_minimum", nilai_minimum])
    writer.writerow(["produk_terlaris", produk_terlaris["produk"]])
    writer.writerow(["jumlah_terjual_terlaris", produk_terlaris["jumlah"]])
    writer.writerow(["produk_omzet_tertinggi",
produk_omzet_tertinggi["produk"]])
    writer.writerow(["produk_omzet_terendah",
produk_omzet_terendah["produk"]])

print("\nFile berhasil disimpan:")
print("- outputs/penjualan_detail.csv")
print("- outputs/ringkasan_penjualan.csv")
```

Kode seperti ini sangat konsisten dengan pola latihan yang ditekankan di referensi: **membaca atau menyiapkan data, menghitung metrik, lalu menulis hasil ke file**, serta mendokumentasikan proses dengan rapi.

4.5 Cara menjalankan di Ubuntu 26.04

Agar pembaca bisa langsung praktik, berikut langkah yang disarankan.

Tahap 1 — Siapkan folder proyek

Masuk ke folder proyek, misalnya:

```
cd ~/projek/python-ai-pemula
```

Kalau struktur folder belum ada, buat:

```
mkdir -p src outputs
```

Tahap 2 — Simpan file program

Simpan kode di atas ke file:

```
src/mini_project_penjualan.py
```

Tahap 3 — Jalankan program

Di terminal Ubuntu 26.04, jalankan:

```
python src/mini_project_penjualan.py
```

Tahap 4 — Periksa hasil

Kalau berhasil, terminal akan menampilkan ringkasan penjualan, lalu dua file akan dibuat:

- `outputs/penjualan_detail.csv`
- `outputs/ringkasan_penjualan.csv`

Tahap 5 — Buka hasil CSV

Pembaca bisa membuka file CSV dengan:

- editor teks
- LibreOffice Calc
- spreadsheet lain
- atau nanti dibaca lagi dengan Python di bab berikutnya

4.6 Contoh keluaran yang diharapkan

Kurang lebih hasil di terminal akan terlihat seperti ini:

```
============================================================
RINGKASAN DATA PENJUALAN TOKO KECIL
============================================================
Produk: Beras 5kg    | Jumlah:  12 | Harga: Rp 74,000 | Omzet: Rp 888,000
Produk: Gula 1kg     | Jumlah:  20 | Harga: Rp 17,000 | Omzet: Rp 340,000
Produk: Minyak 1L    | Jumlah:  15 | Harga: Rp 21,000 | Omzet: Rp 315,000
Produk: Telur 1kg    | Jumlah:  18 | Harga: Rp 28,000 | Omzet: Rp 504,000
Produk: Mi Instan    | Jumlah:  40 | Harga: Rp  3,500 | Omzet: Rp 140,000
------------------------------------------------------------
Total omzet        : Rp2,187,000
Rata-rata omzet    : Rp437,400.00
Nilai maksimum     : Rp888,000
Nilai minimum      : Rp140,000
Produk terlaris    : Mi Instan (40 unit)
Omzet tertinggi    : Beras 5kg (Rp888,000)
Omzet terendah     : Mi Instan (Rp140,000)
============================================================
```

File berhasil disimpan:

- `outputs/penjualan_detail.csv`
- `outputs/ringkasan_penjualan.csv`

Dari contoh ini, pembaca bisa mulai melihat satu pelajaran penting: **produk terlaris belum tentu punya omzet tertinggi**. Mi Instan terjual paling banyak, tetapi karena harga satuannya kecil, omzetnya justru paling rendah. Sebaliknya, Beras 5kg tidak terjual paling banyak, tetapi menghasilkan omzet tertinggi. Inilah contoh awal bagaimana data membantu kita berpikir lebih tajam.

4.7 Bedah kode langkah demi langkah

Agar pembaca tidak hanya menjalankan file, kita bedah logikanya.

1. Menyimpan data awal

```
data_penjualan = [
    {"produk": "Beras 5kg", "jumlah": 12, "harga": 74000},
    ...
]
```

Bagian ini memakai **list of dictionaries**, yaitu pola penting dalam Python dasar.

2. Menghitung omzet

```
def hitung_omzet_produk(jumlah, harga):
    return jumlah * harga
```

Ini adalah fungsi sederhana, tetapi sangat penting. Kita mulai membiasakan bahwa logika yang bisa dipakai berulang sebaiknya dijadikan fungsi.

3. Menambahkan hasil ke setiap item

```
for item in data_penjualan:
    item["omzet"] = hitung_omzet_produk(item["jumlah"], item["harga"])
```

Di sinilah pembaca melihat hubungan antara:

- `for`
- `dict`
- fungsi
- assignment nilai baru

4. Menghitung total dan rata-rata

```
total_omzet = hitung_total_omzet(data_penjualan)
rata_rata_omzet = hitung_rata_rata_omzet(data_penjualan)
```

Ini mengajarkan pembaca bahwa setelah data siap, kita bisa membuat **agregasi**. Dalam dunia analitik, agregasi seperti ini sangat sering dilakukan.

5. Mencari nilai maksimum dan minimum

Alih-alih langsung memakai fungsi canggih, di sini kita membuat fungsi sendiri. Tujuannya agar pembaca benar-benar memahami logikanya.

6. Menulis hasil ke CSV

Bagian ini sangat penting, karena dari sinilah pembaca mulai bergerak dari “program kecil” ke “alat kerja yang menghasilkan output nyata”. Referensi lampiran juga berulang kali menekankan bahwa **membaca data dan menulis hasil** adalah fondasi analisis yang sangat penting.

4.8 Isi file CSV yang dihasilkan

File 1 — penjualan_detail.csv

Isi file ini kira-kira:

```
produk,jumlah,harga,omzet
Beras 5kg,12,74000,888000
Gula 1kg,20,17000,340000
Minyak 1L,15,21000,315000
Telur 1kg,18,28000,504000
Mi Instan,40,3500,140000
```

File 2 — ringkasan_penjualan.csv

Isi file ini kira-kira:

```
metrik,nilai
total_omzet,2187000
rata_rata_omzet,437400.0
nilai_maksimum,888000
nilai_minimum,140000
produk_terlaris,Mi Instan
jumlah_terjual_terlaris,40
produk_omzet_tertinggi,Beras 5kg
produk_omzet_terendah,Mi Instan
```

Dengan dua file ini, pembaca sudah melakukan hal yang sangat penting dalam dunia data:

- menghasilkan **data detail**
- menghasilkan **ringkasan**
- menyimpan hasil dalam format yang bisa dipakai lagi

4.9 Versi lanjutan: membaca data penjualan dari CSV

Kalau pembaca ingin sedikit tantangan tambahan, data tidak harus ditulis langsung di kode. Data bisa dibaca dari file CSV.

Misalnya buat file:

```
data/raw/penjualan.csv
```

Isi file:

```
produk,jumlah,harga
Beras 5kg,12,74000
Gula 1kg,20,17000
Minyak 1L,15,21000
Telur 1kg,18,28000
Mi Instan,40,3500
```

Lalu gunakan program berikut.

```
import csv

def hitung_omzet_produk(jumlah, harga):
    return jumlah * harga

def hitung_total_omzet(data):
    total = 0
    for item in data:
        total += item["omzet"]
```

```
    return total

def hitung_rata_rata_omzet(data):
    if len(data) == 0:
        return 0
    return hitung_total_omzet(data) / len(data)

def cari_produk_terlaris(data):
    produk_terlaris = data[0]
    for item in data:
        if item["jumlah"] > produk_terlaris["jumlah"]:
            produk_terlaris = item
    return produk_terlaris

def cari_produk_omzet_tertinggi(data):
    produk_tertinggi = data[0]
    for item in data:
        if item["omzet"] > produk_tertinggi["omzet"]:
            produk_tertinggi = item
    return produk_tertinggi

def cari_produk_omzet_terendah(data):
    produk_terendah = data[0]
    for item in data:
        if item["omzet"] < produk_terendah["omzet"]:
            produk_terendah = item
    return produk_terendah

data_penjualan = []

with open("data/raw/penjualan.csv", "r", encoding="utf-8") as file:
    reader = csv.DictReader(file)
    for row in reader:
        produk = row["produk"]
        jumlah = int(row["jumlah"])
        harga = int(row["harga"])
        omzet = hitung_omzet_produk(jumlah, harga)

        data_penjualan.append({
            "produk": produk,
            "jumlah": jumlah,
            "harga": harga,
            "omzet": omzet
        })

total_omzet = hitung_total_omzet(data_penjualan)
rata_rata_omzet = hitung_rata_rata_omzet(data_penjualan)
produk_terlaris = cari_produk_terlaris(data_penjualan)
produk_tertinggi = cari_produk_omzet_tertinggi(data_penjualan)
produk_terendah = cari_produk_omzet_terendah(data_penjualan)

print("Total omzet:", total_omzet)
print("Rata-rata omzet:", rata_rata_omzet)
```

```
    print("Produk terlaris:", produk_terlaris["produk"])
    print("Omzet tertinggi:", produk_tertinggi["produk"],
    produk_tertinggi["omzet"])
    print("Omzet terendah:", produk_terendah["produk"],
    produk_terendah["omzet"])
```

Versi ini sangat bagus karena memperkuat keterampilan:

- membaca CSV
- konversi tipe data
- menyusun list of dict
- mengolah data yang datang dari file

Ini sangat sejalan dengan referensi yang menjadikan CSV sebagai salah satu format utama dalam *data workflow* Python.

4.10 Langkah praktik yang disarankan untuk pembaca

Agar mini project ini benar-benar melatih keterampilan, berikut tahap yang disarankan.

Tahap 1 — Jalankan kode apa adanya

- salin kode lengkap
- simpan ke `src/mini_project_penjualan.py`
- jalankan di Ubuntu 26.04
- pastikan hasil muncul dan file CSV terbentuk

Tahap 2 — Pahami logikanya

- identifikasi data awal
- lihat fungsi mana yang menghitung apa
- pahami alur `for`
- perhatikan bagaimana hasil disimpan ke file

Tahap 3 — Ubah datanya

- tambahkan produk baru
- ubah harga
- ubah jumlah penjualan
- jalankan lagi dan lihat perubahan hasil

Tahap 4 — Tambahkan analisis kecil

Coba tambahkan:

- kategori produk
- stok awal
- sisa stok
- persen kontribusi omzet tiap produk

Tahap 5 — Ubah output

- simpan juga ke JSON
- tampilkan format laporan yang lebih rapi
- buat file ringkasan harian dan mingguan

Latihan bertahap seperti ini sangat penting, karena banyak referensi juga mendorong pendekatan *hands-on* dan *step-by-step* agar pembaca benar-benar membangun kompetensi, bukan sekadar menyelesaikan contoh.

4.11 Tantangan pengembangan untuk pembaca

Setelah versi dasar selesai, pembaca bisa mencoba tantangan berikut:

- **Tambahkan validasi input**
  Pastikan `jumlah` dan `harga` tidak negatif.
- **Tambahkan fitur kategori**
  Misalnya: sembako, minuman, makanan instan.
- **Tambahkan perhitungan laba sederhana**
  Simpan harga modal lalu hitung laba per produk.
- **Simpan juga ke JSON**
  Ini bagus untuk melatih struktur data modern.
- **Pisahkan kode ke beberapa fungsi yang lebih kecil**
  Ini latihan menuju kode yang lebih rapi dan modular.
- **Buat versi interaktif**
  Minta pengguna memasukkan data lewat `input()`.

Tantangan seperti ini membuat pembelajaran terasa **fun, menantang, dan membangkitkan rasa ingin mencoba**. Di sinilah pemula mulai naik kelas.

4.12 Refleksi: dari coding ke berpikir data

Inilah bagian paling penting dari mini project ini. Setelah program berjalan, pembaca perlu berhenti sejenak dan bertanya:

Apa sebenarnya yang saya lakukan?

Jawabannya bukan sekadar “menulis Python”. Yang sedang dilakukan pembaca adalah:

- menyusun data
- menentukan ukuran yang penting
- menghitung metrik
- membandingkan nilai
- membuat ringkasan
- menghasilkan keluaran yang bisa dipakai untuk keputusan

Inilah inti dari **berpikir data**.

Dalam coding biasa, seseorang bisa puas hanya karena program berjalan. Tetapi dalam berpikir data, pertanyaan berikutnya adalah:

- angka ini artinya apa?
- produk mana yang benar-benar penting?
- apakah volume penjualan tinggi selalu berarti omzet tinggi?
- apa tindakan yang sebaiknya dilakukan pemilik toko?

Dari hasil mini project ini, misalnya, kita belajar:

- **Mi Instan** paling laku dari sisi jumlah unit
- tetapi **Beras 5kg** paling besar kontribusi omzetnya
- artinya keputusan bisnis tidak cukup hanya melihat “yang paling banyak terjual”
- kita perlu melihat **nilai ekonomi dari penjualan**

Inilah lompatan kecil tetapi sangat penting: dari *coding* ke **analisis**. Dan dari analisis, nanti pembaca akan bergerak ke statistik, visualisasi, dan *machine learning*.

Refleksi seperti ini juga selaras dengan arah outline utama yang menekankan bahwa pembaca tidak boleh berhenti di “menjalankan library”, tetapi harus paham **apa yang sedang dihitung** dan **bagaimana menarik makna dari hasilnya**.

Penutup Bab

Mini Project 1 ini membuktikan bahwa **Python dasar sudah cukup untuk menghasilkan analisis yang berguna**. Dengan memakai variabel, `list`, `dict`, `for`, fungsi, dan file CSV, pembaca bisa membangun alat kecil untuk membaca data penjualan, menghitung metrik penting, dan menyimpan hasil dalam format yang rapi. Ini sangat sesuai dengan posisi mini project pada outline buku utama: sebagai latihan nyata yang menghubungkan fondasi Python dengan cara berpikir data.

Lebih penting lagi, bab ini mengajarkan satu pelajaran besar: **data tidak harus besar untuk menjadi berguna**. Bahkan dari lima produk sederhana, kita sudah bisa melihat pola, membandingkan performa, dan menarik insight awal. Itulah sebabnya mini project seperti ini sangat penting untuk pemula. Ia membangun rasa percaya diri, menunjukkan manfaat Python secara langsung, dan mempersiapkan pembaca untuk bab-bab berikutnya yang lebih kaya, seperti NumPy, Pandas, visualisasi, statistik, dan *machine learning*.

Tantangan kecil untuk pembaca: ubah mini project ini agar bisa menghitung **laba**, bukan hanya omzet. Tambahkan `harga_modal`, lalu hitung produk dengan laba tertinggi. Begitu Anda bisa melakukannya, berarti Anda sudah mulai benar-benar berpikir seperti orang data.

Bagian II — Data Analysis dan Statistik yang Benar-Benar Dipakai

Bab 5. NumPy untuk Data Numerik

Bab ini adalah langkah penting dalam perjalanan dari **Python dasar** menuju **analisis data dan machine learning**. Di outline naskah utama, NumPy ditempatkan sebagai pintu masuk ke dunia komputasi numerik karena pembaca mulai beralih dari *list* Python biasa ke **array numerik yang lebih cepat, lebih ringkas, dan lebih cocok untuk data science**. Outline tersebut juga menegaskan bahwa alur buku harus bergerak secara bertahap: dari Python dasar, lalu ke data numerik, lalu ke statistik, visualisasi, dan akhirnya ke model AI/ML. Karena itu, menguasai NumPy bukan sekadar belajar *library* baru, tetapi belajar **cara berpikir yang lebih numerik, lebih efisien, dan lebih siap untuk machine learning**.

Materi ini juga sangat konsisten dengan buku-buku referensi pendamping. Jain menempatkan NumPy sebagai **fondasi komputasi numerik** di Python, dengan fokus pada **pembuatan array, operasi elemen-per-elemen, indexing, slicing, reshaping, dan fungsi statistik**. Saltz menunjukkan bahwa dalam praktik machine learning, data prediktor sering perlu diubah menjadi **array NumPy dua dimensi** melalui `reshape()` agar bisa dipakai model seperti *Linear Regression*. Sementara beberapa referensi lain menekankan bahwa keunggulan utama NumPy ada pada **array multidimensi, operasi vektorisasi, dan fungsi statistik yang efisien**.

Bab ini disusun agar **mudah, fun, menantang, tetapi tetap membumi**. Fokusnya bukan teori yang terlalu matematis, melainkan bagaimana pembaca benar-benar bisa:

- **membuat array NumPy**
- **melakukan operasi numerik**
- **memakai indexing dan slicing**
- **mengubah bentuk data dengan `reshape()`**
- **memahami operasi vektor dan matriks dasar**
- **menghitung statistik sederhana seperti mean, median, variance, dan standard deviation**

Semua contoh ditulis agar cocok dijalankan di **Ubuntu 26.04** dengan lingkungan Python yang sudah disiapkan di bab sebelumnya.

5.1 Mengapa NumPy penting untuk calon praktisi AI

Sebelum memakai NumPy, pembaca biasanya bekerja dengan `list` Python. Itu cukup baik untuk latihan awal, tetapi ketika data mulai lebih banyak dan perhitungan mulai lebih intensif, `list` punya keterbatasan. NumPy hadir untuk mengatasi masalah itu dengan menyediakan **array multidimensi homogen** yang dirancang untuk **penyimpanan dan komputasi numerik yang efisien**. Jain secara eksplisit menekankan bahwa NumPy arrays lebih cepat dan lebih hemat memori dibanding `list` Python, terutama untuk dataset yang lebih besar.

Dalam praktik AI dan *data science*, NumPy penting karena:

- data sering berbentuk **sekumpulan angka**
- operasi seperti tambah, kali, rata-rata, dan transformasi bentuk data dilakukan berulang
- model machine learning biasanya mengharapkan data dalam bentuk **array** atau **matriks**
- banyak *library* lain, seperti Pandas, SciPy, dan scikit-learn, dibangun di atas fondasi NumPy

Cara paling sederhana untuk memahami perannya adalah ini:
kalau Python dasar mengajarkan kita **cara menulis logika**, maka NumPy mengajarkan kita **cara mengolah angka secara efisien**.

5.2 Menyiapkan NumPy di Ubuntu 26.04

Jika pembaca sudah mengikuti bab penyiapan *ecosystem*, NumPy biasanya sudah terpasang. Namun untuk memastikan, jalankan perintah berikut di terminal Ubuntu 26.04 dari *virtual environment* proyek:

```
pip install numpy
```

Lalu uji dengan membuat file `src/cek_numpy.py`:

```
import numpy as np

print("NumPy berhasil dipanggil")
print("Versi NumPy:", np.__version__)
```

Jalankan:

```
python src/cek_numpy.py
```

Kalau berhasil, pembaca siap masuk ke praktik.

5.3 Array dan operasi numerik

Apa itu array NumPy

Array NumPy adalah struktur data numerik yang dapat menyimpan banyak nilai dengan tipe seragam. Jain menjelaskan bahwa array NumPy bisa dibuat dari `list`, `tuple`, atau *iterable* lain menggunakan `np.array()`, dan bisa berbentuk 1D maupun 2D.

Membuat array 1 dimensi

Buat file `src/numpy_array_dasar.py`:

```
import numpy as np

data_nilai = np.array([80, 85, 90, 75, 88])

print("Array nilai:", data_nilai)
print("Tipe data:", type(data_nilai))
print("Bentuk array:", data_nilai.shape)
```

Contoh keluaran:

```
Array nilai: [80 85 90 75 88]
Tipe data: <class 'numpy.ndarray'>
Bentuk array: (5,)
```

Di sini pembaca mulai melihat hal penting: array NumPy bukan `list`, melainkan objek `ndarray`.

Membuat array 2 dimensi

```
import numpy as np

matriks_nilai = np.array([
    [80, 85, 90],
    [75, 88, 92]
])

print("Array 2D:")
print(matriks_nilai)
print("Shape:", matriks_nilai.shape)
```

Contoh keluaran:

```
Array 2D:
[[80 85 90]
 [75 88 92]]
Shape: (2, 3)
```

Angka `(2, 3)` berarti array punya **2 baris dan 3 kolom**.

Membuat array khusus

NumPy menyediakan beberapa fungsi praktis seperti `zeros`, `ones`, dan `arange`. Ini juga dijelaskan pada referensi Jain sebagai bagian dari pembuatan array dengan sifat khusus.

```
import numpy as np

arr_zeros = np.zeros((2, 3))
arr_ones = np.ones((3, 2))
arr_range = np.arange(0, 10, 2)

print("Array nol:")
print(arr_zeros)

print("\nArray satu:")
print(arr_ones)

print("\nArray rentang:")
print(arr_range)
```

Operasi numerik dasar

Salah satu kekuatan utama NumPy adalah **operasi elemen-per-elemen**. Jain menyebut ini sebagai *vectorization*, salah satu alasan utama NumPy sangat efisien.

```
import numpy as np

a = np.array([1, 2, 3])
b = np.array([4, 5, 6])

print("a + b =", a + b)
print("a - b =", a - b)
print("a * b =", a * b)
print("a / b =", a / b)
```

Contoh keluaran:

```
a + b = [5 7 9]
a - b = [-3 -3 -3]
a * b = [ 4 10 18]
a / b = [0.25 0.4  0.5 ]
```

Operasi dengan skalar

NumPy juga mendukung *broadcasting* sederhana, misalnya mengalikan semua elemen array dengan satu angka. Jain memberi contoh pengalian array dengan skalar sebagai ilustrasi *broadcasting*.

```
import numpy as np

data = np.array([10, 20, 30, 40])

print("Tambah 5:", data + 5)
print("Kali 2:", data * 2)
print("Bagi 10:", data / 10)
```

Ini sangat berguna misalnya saat:

- menaikkan harga semua produk dengan persentase tertentu
- mengubah satuan meter ke sentimeter
- menormalkan data

Langkah praktik yang disarankan

- buat array 1D dari data nilai mahasiswa
- buat array 2D dari dua baris nilai
- coba `np.zeros()`, `np.ones()`, `np.arange()`
- lakukan operasi `+`, `-`, `*`, `/`
- uji operasi dengan skalar

Latihan sederhana ini sangat penting karena menjadi dasar untuk operasi yang lebih kompleks nanti.

5.4 Indexing, slicing, dan `reshape`

Setelah pembaca bisa membuat array, langkah berikutnya adalah **mengambil bagian tertentu dari array** dan **mengubah bentuknya**. Referensi Jain secara khusus menempatkan *indexing, slicing, and reshaping* sebagai inti pembelajaran NumPy.

A. Indexing array 1D

```
import numpy as np

arr = np.array([10, 20, 30, 40, 50])

print("Elemen pertama:", arr[0])
print("Elemen ketiga:", arr[2])
print("Elemen terakhir:", arr[-1])
```

B. Slicing array 1D

```
import numpy as np

arr = np.array([10, 20, 30, 40, 50])

print("Dua elemen pertama:", arr[:2])
print("Elemen indeks 1 sampai 3:", arr[1:4])
print("Mulai indeks 2 sampai akhir:", arr[2:])
```

C. Indexing array 2D

Untuk array dua dimensi, gunakan format `[baris, kolom]`.

```
import numpy as np

arr2d = np.array([
    [1, 2, 3],
    [4, 5, 6],
    [7, 8, 9]
])

print("Elemen baris 1 kolom 2:", arr2d[0, 1])
print("Elemen baris 3 kolom 1:", arr2d[2, 0])
```

D. Slicing array 2D

Jain memberikan contoh pengambilan *subarray* pada array 2D.

```
import numpy as np

arr2d = np.array([
    [1, 2, 3],
    [4, 5, 6],
    [7, 8, 9]
])

print("Dua baris pertama, dua kolom terakhir:")
print(arr2d[0:2, 1:3])
```

Contoh hasil:

```
[[2 3]
 [5 6]]
```

E. Mengapa slicing penting

Dalam pekerjaan data, kita sering perlu:

- mengambil kolom tertentu
- mengambil baris pertama sampai ke-100
- memisahkan fitur dan target
- membagi data latihan dan data uji

Slicing adalah kebiasaan penting yang harus dikuasai sejak awal.

F. reshape(): mengubah bentuk array

Salah satu konsep paling penting di NumPy adalah `reshape()`. Jain menjelaskan bahwa `reshape()` digunakan untuk mengubah bentuk array tanpa mengubah datanya. Saltz juga menunjukkan bahwa dalam *machine learning*, data prediktor tunggal sering harus diubah menjadi **array 2D** memakai `reshape(-1, 1)` karena model mengharapkan matriks fitur, bukan daftar angka biasa.

Contoh dasar:

```
import numpy as np

arr = np.arange(1, 10)
print("Array asli:")
print(arr)

reshaped = arr.reshape(3, 3)
print("\nArray setelah reshape 3x3:")
print(reshaped)
```

Contoh hasil:

```
Array asli:
[1 2 3 4 5 6 7 8 9]
```

Array setelah reshape 3x3:

```
[[1 2 3]
 [4 5 6]
 [7 8 9]]
```

G. Contoh reshape(-1, 1)

Ini sangat penting untuk persiapan *machine learning*.

```
import numpy as np

umur = np.array([18, 19, 20, 21, 22])
X = umur.reshape(-1, 1)

print("Array asli:", umur)
print("Shape asli:", umur.shape)
print("\nSetelah reshape:")
print(X)
print("Shape baru:", X.shape)
```

Hasilnya:

```
Array asli: [18 19 20 21 22]
Shape asli: (5,)
```

Setelah reshape:

```
[[18]
 [19]
 [20]
 [21]
 [22]]
Shape baru: (5, 1)
```

Di sini:

- `-1` berarti NumPy menentukan jumlah baris otomatis
- `1` berarti kita ingin 1 kolom

Langkah praktik yang disarankan

- buat array 1D berisi 10 angka
- ambil elemen pertama, tengah, dan terakhir
- lakukan slicing beberapa bagian
- ubah array 1D menjadi 2D
- coba `reshape(2, 5)` dan `reshape(5, 2)`
- buat contoh `reshape(-1, 1)` untuk data usia atau nilai

5.5 Operasi vektor dan matriks dasar

Setelah paham array, indexing, dan reshape, pembaca siap masuk ke **operasi vektor dan matriks dasar**. Ini penting karena banyak model AI/ML bekerja di atas representasi vektor dan matriks. Referensi NumPy dalam lampiran menekankan bahwa keunggulan NumPy ada pada dukungan array multidimensi dan operasi numerik tingkat tinggi, termasuk yang relevan untuk komputasi matriks.

A. Vektor sebagai array 1D

```
import numpy as np

vektor_a = np.array([1, 2, 3])
vektor_b = np.array([4, 5, 6])

print("Penjumlahan vektor:", vektor_a + vektor_b)
print("Pengurangan vektor:", vektor_a - vektor_b)
```

B. Perkalian elemen-per-elemen

```
import numpy as np

vektor_a = np.array([1, 2, 3])
vektor_b = np.array([4, 5, 6])

print("Perkalian elemen-per-elemen:", vektor_a * vektor_b)
```

Hasil:

```
[ 4 10 18]
```

Ini **bukan** perkalian titik (*dot product*), melainkan perkalian setiap posisi elemen.

C. Dot product

Untuk menghitung hasil kali titik, gunakan `np.dot()` atau operator `@`.

```
import numpy as np

vektor_a = np.array([1, 2, 3])
vektor_b = np.array([4, 5, 6])

dot_hasil = np.dot(vektor_a, vektor_b)
print("Dot product:", dot_hasil)
```

Perhitungannya:

```
1×4 + 2×5 + 3×6 = 32
```

D. Matriks dasar

```
import numpy as np

A = np.array([
    [1, 2],
    [3, 4]
])

B = np.array([
    [5, 6],
    [7, 8]
])

print("A + B:")
print(A + B)

print("\nA - B:")
print(A - B)
```

E. Perkalian elemen matriks

```
import numpy as np

A = np.array([
    [1, 2],
    [3, 4]
])

B = np.array([
    [5, 6],
    [7, 8]
])

print("Perkalian elemen matriks:")
print(A * B)
```

F. Perkalian matriks sesungguhnya

Gunakan `@` atau `np.matmul()`.

```
import numpy as np

A = np.array([
    [1, 2],
    [3, 4]
])

B = np.array([
    [5, 6],
    [7, 8]
])

print("Perkalian matriks:")
print(A @ B)
```

Hasil:

```
[[19 22]
 [43 50]]
```

G. Kenapa ini penting untuk AI/ML

Banyak perhitungan model, terutama regresi, *neural network*, dan transformasi fitur, pada dasarnya adalah operasi vektor dan matriks. Karena itu, walaupun pembaca belum masuk ke matematika yang dalam, memahami operasi dasar ini akan sangat membantu saat nanti melihat istilah seperti:

- matriks fitur `X`
- vektor target `y`
- koefisien model
- transformasi data

Contoh kecil yang relevan

Misalnya satu siswa punya tiga nilai komponen: tugas, UTS, UAS. Kita ingin menghitung nilai akhir dengan bobot tertentu.

```
import numpy as np

nilai = np.array([80, 75, 90])
bobot = np.array([0.3, 0.3, 0.4])

nilai_akhir = np.dot(nilai, bobot)
print("Nilai akhir:", nilai_akhir)
```

Ini contoh yang sangat bagus karena menunjukkan bahwa operasi vektor bukan hal abstrak. Ia langsung berguna dalam kasus nyata.

Langkah praktik yang disarankan

- buat dua vektor, lalu jumlahkan
- hitung `dot product`
- buat dua matriks kecil
- bandingkan `A * B` dan `A @ B`
- coba kasus bobot nilai atau omzet dengan vektor

5.6 Statistik sederhana dengan NumPy

Salah satu alasan NumPy sangat berguna adalah karena ia menyediakan fungsi statistik bawaan yang efisien. Jain menekankan fungsi seperti `mean`, `median`, dan `variance`, termasuk perhitungan menurut sumbu baris atau kolom. Buku statistik berbasis Python juga memberi banyak contoh penggunaan `np.mean()`, `np.std()`, dan fungsi lain untuk analisis kuantitatif praktis.

A. Mean, median, variance, standard deviation

```
import numpy as np

data = np.array([70, 75, 80, 85, 90])

print("Mean:", np.mean(data))
print("Median:", np.median(data))
print("Variance:", np.var(data))
print("Standard deviation:", np.std(data))
print("Minimum:", np.min(data))
print("Maximum:", np.max(data))
print("Jumlah total:", np.sum(data))
```

B. Makna intuitif

- **Mean**: rata-rata
- **Median**: nilai tengah
- **Variance**: ukuran sebaran data
- **Standard deviation**: seberapa jauh data menyebar dari rata-rata
- **Min/Max**: nilai terkecil dan terbesar
- **Sum**: total semua nilai

C. Statistik pada array 2D berdasarkan sumbu

Jain menunjukkan bahwa pada array 2D, kita bisa menghitung rata-rata menurut baris (`axis=1`) atau kolom (`axis=0`).

```
import numpy as np

nilai_kelas = np.array([
    [80, 85, 90],
    [75, 88, 92]
])

print("Mean seluruh array:", np.mean(nilai_kelas))
print("Mean per kolom:", np.mean(nilai_kelas, axis=0))
print("Mean per baris:", np.mean(nilai_kelas, axis=1))
```

Hasil kira-kira:

```
Mean seluruh array: 85.0
Mean per kolom: [77.5 86.5 91. ]
Mean per baris: [85.         85.        ]
```

D. Contoh praktis: analisis penjualan bulanan

```
import numpy as np

penjualan = np.array([5700, 6200, 6500, 5980, 5800,
6480])

mean_penjualan = np.mean(penjualan)
std_penjualan = np.std(penjualan)
minimum = np.min(penjualan)
maksimum = np.max(penjualan)

print("Rata-rata penjualan:", mean_penjualan)
print("Standar deviasi:", std_penjualan)
print("Penjualan minimum:", minimum)
print("Penjualan maksimum:", maksimum)
```

Contoh seperti ini juga muncul dalam buku statistik berbasis Python untuk mengukur kestabilan penjualan dan variasi data menggunakan NumPy.

E. ddof dan simpangan baku sampel

Untuk pemula, cukup pahami bahwa:

- `np.std(data)` default menghitung standar deviasi populasi
- `np.std(data, ddof=1)` menghitung standar deviasi sampel

Buku statistik Python memberikan contoh jelas penggunaan `ddof=1` ketika menghitung standar deviasi sampel agar estimasinya lebih tepat untuk data sampel.

Contoh:

```
import numpy as np

data = np.array([49, 51, 46, 54, 50, 50])

print("Std populasi:", np.std(data))
print("Std sampel :", np.std(data, ddof=1))
```

Untuk buku pemula, penjelasannya cukup seperti ini:
kalau data adalah seluruh populasi, gunakan default; kalau data adalah sampel dari populasi yang lebih besar, sering dipakai `ddof=1`.

F. Contoh mini analisis lengkap

```
import numpy as np

nilai = np.array([82, 76, 91, 88, 79, 95, 84])

print("Data nilai:", nilai)
print("Jumlah data:", len(nilai))
print("Rata-rata:", np.mean(nilai))
print("Median:", np.median(nilai))
print("Nilai minimum:", np.min(nilai))
print("Nilai maksimum:", np.max(nilai))
print("Variance:", np.var(nilai))
print("Std dev:", np.std(nilai))
```

Ini adalah contoh yang bagus untuk menunjukkan bagaimana NumPy membantu kita bergerak dari “sekadar angka” menjadi **ringkasan statistik yang bermakna**.

Langkah praktik yang disarankan

- buat array nilai mahasiswa
- hitung mean, median, min, max
- hitung variance dan standard deviation
- buat array 2D nilai dua kelas
- hitung mean per baris dan per kolom
- bandingkan `np.std(data)` dan `np.std(data, ddof=1)`

5.7 Source code lengkap: latihan terpadu NumPy

Agar semua konsep di bab ini menyatu, berikut satu **program lengkap** yang memadukan array, indexing, slicing, reshape, operasi vektor/matriks, dan statistik sederhana.

Simpan sebagai `src/numpy_latihan_terpadu.py`:

```
import numpy as np

print("=" * 60)
print("LATIHAN TERPADU NUMPY")
print("=" * 60)

# 1. Membuat array 1D
nilai = np.array([80, 85, 90, 75, 88, 92, 78])
print("\n1. Array nilai:")
print(nilai)
print("Shape:", nilai.shape)

# 2. Indexing dan slicing
print("\n2. Indexing dan slicing:")
print("Elemen pertama :", nilai[0])
print("Elemen terakhir:", nilai[-1])
print("Tiga elemen awal:", nilai[:3])
print("Elemen indeks 2 sampai 5:", nilai[2:6])

# 3. Operasi numerik
print("\n3. Operasi numerik:")
print("Nilai + 5 :", nilai + 5)
print("Nilai * 2 :", nilai * 2)

# 4. Reshape
print("\n4. Reshape ke 2D:")
nilai_2d = nilai[:6].reshape(2, 3)
print(nilai_2d)
print("Shape baru:", nilai_2d.shape)

# 5. Operasi vektor
print("\n5. Operasi vektor:")
bobot = np.array([0.3, 0.3, 0.4])
nilai_siswa = np.array([80, 75, 90])
nilai_akhir = np.dot(nilai_siswa, bobot)
```

```
print("Nilai siswa:", nilai_siswa)
print("Bobot      :", bobot)
print("Nilai akhir:", nilai_akhir)

# 6. Operasi matriks
print("\n6. Operasi matriks:")
A = np.array([[1, 2], [3, 4]])
B = np.array([[5, 6], [7, 8]])
print("A + B:")
print(A + B)
print("A * B (elemen-per-elemen):")
print(A * B)
print("A @ B (perkalian matriks):")
print(A @ B)

# 7. Statistik sederhana
print("\n7. Statistik sederhana:")
print("Mean   :", np.mean(nilai))
print("Median :", np.median(nilai))
print("Min    :", np.min(nilai))
print("Max    :", np.max(nilai))
print("Var    :", np.var(nilai))
print("Std    :", np.std(nilai))

# 8. Statistik axis pada array 2D
print("\n8. Statistik pada array 2D:")
kelas = np.array([
    [80, 85, 90],
    [75, 88, 92],
    [78, 81, 84]
])

print("Array kelas:")
print(kelas)
print("Mean per kolom:", np.mean(kelas, axis=0))
print("Mean per baris:", np.mean(kelas, axis=1))

print("\nSelesai. Anda sudah memakai NumPy untuk komputasi numerik
dasar.")
print("=" * 60)
```

Jalankan dengan:

```
python src/numpy_latihan_terpadu.py
```

Program ini sangat baik sebagai latihan integratif sebelum masuk ke Pandas dan analisis data tabular.

5.8 Kesalahan umum pemula

Saat mulai memakai NumPy, ada beberapa jebakan umum.

1. Mengira array NumPy sama dengan list Python

Padahal NumPy array mendukung operasi vektorisasi yang berbeda dari list.

```
# List Python
a = [1, 2, 3]
# a + 2  -> error

# NumPy array
import numpy as np
b = np.array([1, 2, 3])
print(b + 2)   # berhasil
```

2. Salah memahami shape

Banyak error muncul karena pembaca tidak sadar apakah data berbentuk:

- `(5,)`
- `(5, 1)`
- `(1, 5)`

Padahal bentuk array sangat penting, terutama menjelang machine learning.

3. Bingung antara * dan @

- `*` untuk perkalian elemen-per-elemen
- `@` untuk perkalian matriks

4. Salah memilih axis

Pada array 2D:

- `axis=0` biasanya berarti **per kolom**
- `axis=1` biasanya berarti **per baris**

Ini perlu sering dilatih agar menjadi intuitif.

5.9 Refleksi: dari angka biasa ke cara berpikir numerik

Bab ini sesungguhnya bukan hanya tentang belajar NumPy. Yang lebih penting adalah pembaca mulai masuk ke **cara berpikir numerik yang lebih sistematis**. Dengan NumPy, pembaca belajar bahwa data tidak harus selalu diperlakukan sebagai daftar item satu per satu. Data bisa dilihat sebagai **vektor**, **matriks**, dan **struktur numerik** yang bisa dihitung secara cepat dan efisien.

Ini adalah langkah penting menuju *machine learning*. Nanti, ketika pembaca melihat:

- `X` sebagai matriks fitur
- `y` sebagai vektor target
- `reshape(-1, 1)` sebelum *fit* model
- mean, variance, dan standard deviation untuk memahami sebaran data

semua itu tidak lagi terasa asing, karena fondasinya sudah dibangun di bab ini. Saltz bahkan menunjukkan secara langsung bahwa untuk *linear modeling*, daftar angka harus diubah ke **array NumPy dua dimensi** melalui `reshape()` agar sesuai dengan ekspektasi model.

Dengan kata lain, NumPy membantu pembaca naik kelas:
dari **menulis Python** menjadi **mengolah data numerik secara serius**.

Penutup Bab

NumPy adalah **fondasi komputasi numerik** dalam ekosistem Python. Melalui bab ini, pembaca telah belajar membuat array, melakukan operasi numerik, memakai indexing dan slicing, mengubah bentuk data dengan `reshape()`, menjalankan operasi vektor dan matriks dasar, serta menghitung statistik sederhana seperti mean, median, variance, dan standard deviation. Semua kemampuan ini sangat sejalan dengan peran NumPy di berbagai referensi pendamping: sebagai dasar untuk data analysis, statistik, dan persiapan machine learning.

Pesan terpenting dari bab ini sederhana: **kalau Python dasar adalah bahasa kerja Anda, maka NumPy adalah mesin hitung cerdas Anda**. Begitu pembaca nyaman dengan NumPy, langkah ke Pandas, visualisasi, statistik yang lebih kaya, dan *machine learning* akan terasa jauh lebih alami.

Tantangan kecil untuk pembaca: buat satu file Python yang menyimpan data penjualan bulanan dalam array NumPy, lalu hitung **total, rata-rata, minimum, maksimum, dan standar deviasi**, kemudian ubah data tersebut menjadi array 2D dengan `reshape(3, 4)`. Begitu latihan ini berhasil, Anda sudah benar-benar mulai berpikir seperti praktisi data.

Bab 6. Pandas untuk Data Nyata

Bab ini adalah titik ketika pembaca mulai merasakan bahwa Python tidak lagi hanya menjadi alat belajar sintaks, tetapi berubah menjadi **alat kerja untuk mengolah data nyata**. Di outline naskah utama, Pandas ditempatkan sebagai **"DataFrame sebagai Excel yang bisa diprogram"**, dengan fokus pada **Series, DataFrame, membaca CSV/Excel, memilih kolom dan baris, filtering, sorting, grouping/agregasi, dan merge/join dasar**. Penempatan ini sangat tepat karena setelah pembaca mengenal Python dasar dan NumPy, langkah alami berikutnya adalah bekerja dengan data tabular yang benar-benar mirip kondisi di lapangan.

Materi ini juga sangat kuat didukung dokumen referensi. Jain menjelaskan bahwa Pandas adalah salah satu *library* paling penting untuk **data manipulation**, dengan dua struktur utamanya, yaitu **Series** dan **DataFrame**, serta kemampuan membaca data dari berbagai sumber, memfilter, mengurutkan, mengelompokkan, dan menggabungkan dataset. Referensi lain juga menegaskan bahwa Pandas membuat pekerjaan yang di Excel sering terasa manual menjadi **lebih singkat, lebih dapat diulang, dan lebih kuat**.

Bab ini disusun agar **mudah diikuti oleh mahasiswa dan praktisi**, namun tetap menantang dan inspiratif. Fokusnya adalah membuat pembaca bisa:

- **memahami perbedaan Series dan DataFrame**
- **membaca file CSV dan Excel**
- **memilih kolom dan baris**
- **memfilter dan mengurutkan data**
- **mengelompokkan data dengan `groupby()`**
- **menggabungkan beberapa tabel dengan `merge()` atau `join()`**
- **membuat ringkasan data cepat untuk memahami isi dataset**

Semua contoh menggunakan **source code Python lengkap** dan bisa dijalankan di **Ubuntu 26.04**.

6.1 Mengapa Pandas penting untuk calon praktisi AI

Dalam pekerjaan nyata, data jarang datang dalam bentuk array angka rapi seperti contoh NumPy. Data sering berbentuk tabel: ada kolom nama, umur, kota, nilai, transaksi, harga, tanggal, kategori, dan lain-lain. Untuk kondisi seperti inilah Pandas menjadi sangat penting. Jain menjelaskan bahwa Pandas menyediakan dua struktur data utama, yakni **Series** yang satu dimensi dan **DataFrame** yang dua dimensi, sehingga sangat cocok untuk data tabular.

Kalau NumPy unggul untuk komputasi numerik, maka Pandas unggul untuk:

- membaca file data dari CSV atau Excel
- memilih kolom tertentu
- memfilter baris berdasarkan kondisi
- mengurutkan tabel
- mengelompokkan data dan menghitung agregat
- menggabungkan beberapa tabel menjadi satu

Bagi pembaca yang familiar dengan spreadsheet, cara paling mudah memahami Pandas adalah ini:

> **DataFrame adalah seperti lembar Excel, tetapi bisa diprogram, diulang, dan diautomasi.** Referensi Python Fusion bahkan secara eksplisit menggambarkan DataFrame sebagai versi worksheet yang lebih kuat dan fleksibel.

Dalam konteks AI/ML, Pandas penting karena sebelum melatih model, kita hampir selalu harus:

- membaca dataset
- memeriksa kolom
- memilih fitur
- membersihkan data
- membuat ringkasan awal
- memahami distribusi sederhana

Karena itu, menguasai Pandas adalah **langkah wajib** sebelum melangkah ke EDA, statistik, dan machine learning.

6.2 Menyiapkan Pandas di Ubuntu 26.04

Kalau pembaca sudah mengikuti bab penyiapan *ecosystem*, kemungkinan Pandas sudah terpasang. Namun untuk memastikan, jalankan:

```
pip install pandas openpyxl
```

Kenapa `openpyxl` juga dipasang? Karena untuk membaca file Excel `.xlsx`, Pandas biasanya membutuhkan *engine* tambahan seperti `openpyxl`.

Lalu buat file `src/cek_pandas.py`:

```
import pandas as pd

print("Pandas berhasil dipanggil")
print("Versi Pandas:", pd.__version__)
```

Jalankan:

```
python src/cek_pandas.py
```

Kalau berhasil, pembaca siap masuk ke praktik.

6.3 Series dan DataFrame

Apa itu Series

Menurut Jain, **Series** adalah struktur data satu dimensi yang bisa menampung berbagai tipe data dan memiliki indeks label untuk memudahkan akses. Ia mirip seperti **satu kolom di spreadsheet** atau satu kolom di tabel SQL.

Contoh Series

Buat file `src/pandas_series_dasar.py`:

```
import pandas as pd

nilai = pd.Series([80, 85, 90, 75], index=["Andi", "Budi",
"Citra", "Dina"])

print("Series nilai:")
print(nilai)

print("\nNilai Budi:", nilai["Budi"])
print("Rata-rata:", nilai.mean())
```

Contoh keluaran:

```
Series nilai:
Andi     80
Budi     85
Citra    90
Dina     75
dtype: int64

Nilai Budi: 85
Rata-rata: 82.5
```

Series cocok dipakai ketika data hanya punya satu dimensi utama, misalnya:

- daftar nilai mahasiswa
- daftar harga produk
- jumlah penjualan per bulan

Apa itu DataFrame

Masih menurut Jain, **DataFrame** adalah struktur data dua dimensi seperti tabel, dengan baris dan kolom. Setiap kolom pada DataFrame sebenarnya adalah sebuah Series, dan DataFrame dapat menampung **tipe data berbeda antar kolom**.

Contoh DataFrame

```
import pandas as pd

data = {
    "nama": ["Andi", "Budi", "Citra"],
    "umur": [20, 21, 19],
    "kota": ["Jakarta", "Bandung", "Depok"]
}

df = pd.DataFrame(data)

print(df)
```

Contoh keluaran:

```
   nama  umur      kota
0   Andi    20   Jakarta
1   Budi    21   Bandung
2  Citra    19     Depok
```

Perbedaan Series dan DataFrame

Secara praktis:

- **Series**: satu dimensi, seperti satu kolom
- **DataFrame**: dua dimensi, seperti tabel lengkap
- **Series** cocok untuk satu jenis data
- **DataFrame** cocok untuk data nyata yang punya banyak kolom sekaligus

Contoh lengkap: Series dan DataFrame sekaligus

```
import pandas as pd

# Series
penjualan = pd.Series([120, 150, 170], index=["Jan", "Feb", "Mar"])
print("=== SERIES ===")
print(penjualan)

# DataFrame
df_penjualan = pd.DataFrame({
    "bulan": ["Jan", "Feb", "Mar"],
    "penjualan": [120, 150, 170],
    "biaya": [80, 95, 110]
})

print("\n=== DATAFRAME ===")
print(df_penjualan)
```

Langkah praktik yang disarankan

- buat satu Series berisi nilai 5 mahasiswa
- buat satu DataFrame dengan kolom nama, nilai, dan kota
- akses satu elemen Series
- hitung mean dari Series
- tampilkan kolom tertentu dari DataFrame

6.4 Membaca CSV dan Excel

Salah satu kekuatan terbesar Pandas adalah membaca data dari file dengan sangat mudah. Jain secara eksplisit menyebut `read_csv()` dan `read_excel()` sebagai fungsi utama untuk memuat data dari CSV dan Excel ke DataFrame. Referensi lain juga menekankan bahwa CSV adalah format yang sangat umum, dan Excel tetap sangat penting dalam pekerjaan bisnis dan analitik.

A. Membaca CSV

Buat file `data/raw/mahasiswa.csv`:

```
nama,nilai,kota
Andi,88,Jakarta
Budi,75,Bandung
Citra,93,Depok
Dina,81,Jakarta
Eka,77,Bandung
```

Lalu baca dengan Pandas:

```
import pandas as pd

df = pd.read_csv("data/raw/mahasiswa.csv")

print(df)
print("\nLima baris pertama:")
print(df.head())
```

B. Membaca Excel

Misalnya ada file `data/raw/mahasiswa.xlsx`, maka:

```
import pandas as pd

df = pd.read_excel("data/raw/mahasiswa.xlsx",
sheet_name="Sheet1")

print(df.head())
```

Jain menegaskan bahwa `read_excel()` dapat membaca file Excel dengan memilih *sheet* tertentu melalui parameter `sheet_name`. Referensi lain juga menunjukkan bahwa Pandas bisa membaca satu *sheet* tertentu maupun semua *sheet* sekaligus.

C. Membaca semua sheet Excel

```
import pandas as pd

semua_sheet = pd.read_excel("data/raw/laporan.xlsx",
sheet_name=None)

print(semua_sheet.keys())
print("\nIsi sheet Penjualan:")
print(semua_sheet["Penjualan"].head())
```

D. Melihat struktur data setelah dibaca

Setelah file dibaca, biasakan langsung cek:

- `head()`
- `shape`
- `columns`
- `dtypes`
- `info()`

Contoh:

```
import pandas as pd

df = pd.read_csv("data/raw/mahasiswa.csv")

print("Shape:", df.shape)
print("Kolom:", df.columns.tolist())
print("\nTipe data:")
print(df.dtypes)
print("\nInfo DataFrame:")
print(df.info())
```

Ini sangat penting, karena dalam dunia data, membaca file saja tidak cukup. Kita harus segera memahami bentuk awal dataset.

E. Menulis kembali ke CSV atau Excel

Pandas juga memudahkan ekspor hasil. Jain menjelaskan penggunaan `to_csv()` dan `to_excel()`, sedangkan referensi lain menunjukkan bahwa ekspor ke Excel bisa dibuat rapi dan profesional.

```
import pandas as pd

df = pd.read_csv("data/raw/mahasiswa.csv")

df.to_csv("outputs/mahasiswa_bersih.csv", index=False)
df.to_excel("outputs/mahasiswa_bersih.xlsx",
index=False)
```

Langkah praktik yang disarankan

- buat satu file CSV kecil
- baca dengan `pd.read_csv()`
- lihat `head()`, `shape`, `columns`, `dtypes`
- simpan ulang ke file lain dengan `to_csv()`
- kalau ada file Excel, coba `pd.read_excel()`

6.5 Memilih kolom dan baris, filter, dan sort

Setelah data masuk ke DataFrame, langkah berikutnya adalah **memilih bagian yang relevan**. Inilah inti manipulasi data. Jain secara khusus menekankan *filtering* dan *sorting* sebagai operasi kunci di Pandas.

A. Memilih kolom

```
import pandas as pd

df = pd.read_csv("data/raw/mahasiswa.csv")

print(df["nama"])
print(df[["nama", "nilai"]])
```

B. Memilih baris dengan loc

```
import pandas as pd

df = pd.read_csv("data/raw/mahasiswa.csv")

print(df.loc[0])
print(df.loc[0:2, ["nama", "nilai"]])
```

C. Memilih baris dengan iloc

`iloc` memakai posisi indeks numerik.

```
import pandas as pd

df = pd.read_csv("data/raw/mahasiswa.csv")

print(df.iloc[0])
print(df.iloc[0:3, 0:2])
```

D. Filtering berdasarkan kondisi

Menurut Jain, filtering dilakukan dengan kondisi logika pada kolom. Contoh yang diberikan adalah memfilter baris ketika usia lebih dari 30. Prinsip yang sama bisa dipakai untuk nilai, kota, kategori, dan sebagainya.

```
import pandas as pd

df = pd.read_csv("data/raw/mahasiswa.csv")

nilai_tinggi = df[df["nilai"] >= 80]
print(nilai_tinggi)
```

E. Filter dengan lebih dari satu kondisi

```
import pandas as pd

df = pd.read_csv("data/raw/mahasiswa.csv")

hasil = df[(df["nilai"] >= 80) & (df["kota"] == "Jakarta")]
print(hasil)
```

F. Mengurutkan data

`sort_values()` dipakai untuk mengurutkan DataFrame.

```
import pandas as pd

df = pd.read_csv("data/raw/mahasiswa.csv")

urut_nilai = df.sort_values(by="nilai", ascending=False)
print(urut_nilai)
```

Jain mencontohkan `sort_values(by='Age', ascending=False)` untuk pengurutan menurun. Pola ini identik untuk kasus lain seperti nilai, omzet, atau penjualan.

G. Contoh lengkap filter dan sort

```
import pandas as pd

df = pd.read_csv("data/raw/mahasiswa.csv")

# Filter nilai >= 80
df_filter = df[df["nilai"] >= 80]

# Urutkan dari tertinggi ke terendah
df_urut = df_filter.sort_values(by="nilai", ascending=False)

print(df_urut)
```

Langkah praktik yang disarankan

- pilih satu kolom dari DataFrame
- pilih dua kolom
- ambil 3 baris pertama
- filter data dengan nilai >= 80
- filter data dari kota tertentu
- urutkan berdasarkan nilai menurun

6.6 `groupby()` dan agregasi

Setelah bisa membaca, memilih, dan memfilter data, pembaca siap ke tahap yang sangat penting: **mengelompokkan data dan menghitung ringkasan**. Outline utama menyebut bagian ini sebagai **grouping dan agregasi**, dan Jain juga menegaskan bahwa `groupby()` adalah alat utama untuk menghitung agregat berdasarkan satu atau lebih kolom.

A. Konsep groupby()

Bayangkan kita punya data penjualan berikut:

```
import pandas as pd

df = pd.DataFrame({
    "kota": ["Jakarta", "Bandung", "Jakarta", "Depok", "Bandung"],
    "produk": ["A", "A", "B", "A", "B"],
    "penjualan": [100, 80, 120, 90, 110]
})

print(df)
```

Kalau kita ingin tahu **total penjualan per kota**, kita bisa pakai `groupby()`.

```
import pandas as pd

df = pd.DataFrame({
    "kota": ["Jakarta", "Bandung", "Jakarta", "Depok", "Bandung"],
    "produk": ["A", "A", "B", "A", "B"],
    "penjualan": [100, 80, 120, 90, 110]
})

group_kota = df.groupby("kota")["penjualan"].sum()
print(group_kota)
```

B. Rata-rata per grup

Jain memberi contoh `df.groupby('City')['Age'].mean()` untuk menghitung rata-rata umur per kota. Prinsip yang sama bisa dipakai untuk berbagai metrik lain.

```
import pandas as pd

df = pd.DataFrame({
    "kota": ["Jakarta", "Bandung", "Jakarta", "Depok", "Bandung"],
    "nilai": [88, 75, 93, 81, 77]
})

rata_kota = df.groupby("kota")["nilai"].mean()
print(rata_kota)
```

C. Banyak agregasi sekaligus

```
import pandas as pd

df = pd.DataFrame({
    "kota": ["Jakarta", "Bandung", "Jakarta", "Depok", "Bandung"],
    "nilai": [88, 75, 93, 81, 77]
})

ringkasan = df.groupby("kota")["nilai"].agg(["count", "mean", "min",
"max"])
print(ringkasan)
```

D. Groupby lebih dari satu kolom

```
import pandas as pd

df = pd.DataFrame({
    "kota": ["Jakarta", "Bandung", "Jakarta", "Depok", "Bandung"],
    "jurusan": ["TI", "TI", "SI", "TI", "SI"],
    "nilai": [88, 75, 93, 81, 77]
})

hasil = df.groupby(["kota", "jurusan"])["nilai"].mean()
print(hasil)
```

E. `reset_index()` agar lebih rapi

Hasil `groupby()` sering lebih enak dibaca jika dikembalikan ke bentuk DataFrame.

```
import pandas as pd

df = pd.DataFrame({
    "kota": ["Jakarta", "Bandung", "Jakarta", "Depok", "Bandung"],
    "nilai": [88, 75, 93, 81, 77]
})

hasil = df.groupby("kota")["nilai"].mean().reset_index()
print(hasil)
```

Langkah praktik yang disarankan

- buat DataFrame penjualan atau nilai
- hitung total per kota
- hitung rata-rata per kota
- hitung min dan max
- coba `groupby()` dengan dua kolom
- gunakan `reset_index()`

6.7 Merge dan join

Dalam data nyata, sering kali informasi tidak berada dalam satu tabel saja. Misalnya:

- tabel mahasiswa berisi `id`, `nama`, `jurusan`
- tabel nilai berisi `id`, `nilai`

Untuk menggabungkan dua tabel seperti ini, Pandas menyediakan `merge()` dan `join()`. Jain menjelaskan bahwa `merge()` menggabungkan DataFrame berdasarkan *common key*, sedangkan `join()` umumnya berdasarkan indeks. Referensi Python Fusion juga menekankan bahwa `merge()` sangat kuat, mirip *VLOOKUP* di Excel, tetapi lebih fleksibel.

A. Contoh `merge()`

```
import pandas as pd

df_mahasiswa = pd.DataFrame({
    "id": [1, 2, 3],
    "nama": ["Andi", "Budi", "Citra"]
})

df_nilai = pd.DataFrame({
    "id": [1, 2, 3],
    "nilai": [88, 75, 93]
})

df_gabung = pd.merge(df_mahasiswa, df_nilai, on="id")
print(df_gabung)
```

Hasil:

```
  id   nama  nilai
0   1   Andi     88
1   2   Budi     75
2   3  Citra     93
```

B. how= pada merge

Kita bisa menentukan jenis gabungan:

- `inner`
- `left`
- `right`
- `outer`

Contoh:

```
import pandas as pd

df1 = pd.DataFrame({
    "id": [1, 2, 3],
    "nama": ["Andi", "Budi", "Citra"]
})

df2 = pd.DataFrame({
    "id": [2, 3, 4],
    "nilai": [75, 93, 85]
})

print("INNER JOIN")
print(pd.merge(df1, df2, on="id", how="inner"))
```

```
print("\nLEFT JOIN")
print(pd.merge(df1, df2, on="id", how="left"))

print("\nOUTER JOIN")
print(pd.merge(df1, df2, on="id", how="outer"))
```

C. Contoh join()

Menurut Jain, `join()` sering dipakai ketika DataFrame sudah memakai indeks sebagai kunci.

```
import pandas as pd

df1 = pd.DataFrame({
    "nama": ["Andi", "Budi", "Citra"]
}, index=[1, 2, 3])

df2 = pd.DataFrame({
    "nilai": [88, 75, 93]
}, index=[1, 2, 3])

hasil = df1.join(df2, how="inner")
print(hasil)
```

D. Kapan memakai merge, kapan join

Secara praktis:

- gunakan **`merge()`** jika penggabungan berdasarkan kolom kunci
- gunakan **`join()`** jika penggabungan lebih alami berdasarkan indeks

E. Contoh yang dekat dengan dunia nyata

```
import pandas as pd

penjualan = pd.DataFrame({
    "tanggal": ["2026-01-01", "2026-01-02", "2026-01-03"],
    "omzet": [120000, 150000, 175000]
})

stok = pd.DataFrame({
    "tanggal": ["2026-01-01", "2026-01-02", "2026-01-03"],
    "stok_akhir": [50, 45, 39]
})

gabung = pd.merge(penjualan, stok, on="tanggal")
print(gabung)
```

Ini menunjukkan bagaimana dua sumber data bisa disatukan menjadi satu tabel analisis.

Langkah praktik yang disarankan

- buat dua DataFrame dengan kolom kunci sama
- gabungkan dengan `merge()`
- coba `inner`, `left`, dan `outer`
- buat dua DataFrame dengan indeks sama
- gabungkan dengan `join()`

6.8 Ringkasan data cepat

Salah satu kebiasaan terbaik seorang praktisi data adalah **tidak langsung menganalisis terlalu jauh sebelum melihat ringkasan awal dataset**. Inilah yang saya sebut *quick profiling*. Walaupun outline utama menyebut bagian ini sebagai “ringkasan data cepat”, praktiknya sangat penting sebagai tahap awal EDA. Beberapa fungsi Pandas yang paling berguna adalah:

- `head()`
- `tail()`
- `shape`
- `columns`
- `dtypes`
- `info()`
- `describe()`
- `value_counts()`

A. head() dan tail()

```
import pandas as pd

df = pd.read_csv("data/raw/mahasiswa.csv")

print(df.head())
print(df.tail())
```

B. shape dan columns

```
import pandas as pd

df = pd.read_csv("data/raw/mahasiswa.csv")

print("Jumlah baris dan kolom:", df.shape)
print("Daftar kolom:", df.columns.tolist())
```

C. `dtypes` dan `info()`

```
import pandas as pd

df = pd.read_csv("data/raw/mahasiswa.csv")

print(df.dtypes)
print(df.info())
```

Fungsi ini sangat membantu untuk melihat apakah kolom numerik terbaca sebagai angka atau justru sebagai teks.

D. describe()

```
import pandas as pd

df = pd.read_csv("data/raw/mahasiswa.csv")

print(df.describe())
```

`describe()` memberi ringkasan statistik cepat untuk kolom numerik, seperti count, mean, std, min, max, dan kuartil.

E. value_counts()

Untuk data kategori, `value_counts()` sangat berguna.

```
import pandas as pd

df = pd.read_csv("data/raw/mahasiswa.csv")

print(df["kota"].value_counts())
```

F. Contoh alur ringkasan cepat

```
import pandas as pd

df = pd.read_csv("data/raw/mahasiswa.csv")

print("=== HEAD ===")
print(df.head())

print("\n=== SHAPE ===")
print(df.shape)

print("\n=== DTYPES ===")
print(df.dtypes)

print("\n=== INFO ===")
print(df.info())
```

```
print("\n=== DESCRIBE ===")
print(df.describe())

print("\n=== VALUE COUNTS KOTA ===")
print(df["kota"].value_counts())
```

Inilah langkah awal yang sangat sehat sebelum lanjut ke cleaning, visualisasi, atau modeling.

6.9 Source code Python lengkap: latihan terpadu Pandas

Berikut **program lengkap** yang memadukan pembuatan DataFrame, membaca CSV, filtering, sorting, `groupby()`, `merge()`, dan ringkasan cepat.

Simpan sebagai `src/pandas_latihan_terpadu.py`:

```
import pandas as pd

print("=" * 60)
print("LATIHAN TERPADU PANDAS")
print("=" * 60)

# 1. Membuat DataFrame awal
df_mahasiswa = pd.DataFrame({
    "id": [1, 2, 3, 4, 5],
    "nama": ["Andi", "Budi", "Citra", "Dina", "Eka"],
    "nilai": [88, 75, 93, 81, 77],
    "kota": ["Jakarta", "Bandung", "Depok", "Jakarta",
"Bandung"]
})

print("\n1. DataFrame awal:")
print(df_mahasiswa)

# 2. Ringkasan cepat
print("\n2. Ringkasan cepat:")
print("Shape:", df_mahasiswa.shape)
print("Kolom:", df_mahasiswa.columns.tolist())
print("Tipe data:")
print(df_mahasiswa.dtypes)
print("\nDescribe:")
print(df_mahasiswa.describe())

# 3. Memilih kolom
print("\n3. Memilih kolom nama dan nilai:")
print(df_mahasiswa[["nama", "nilai"]])
```

```
# 4. Filtering
print("\n4. Filter nilai >= 80:")
df_filter = df_mahasiswa[df_mahasiswa["nilai"] >= 80]
print(df_filter)

# 5. Sorting
print("\n5. Sort nilai tertinggi ke terendah:")
df_sort = df_mahasiswa.sort_values(by="nilai",
ascending=False)
print(df_sort)

# 6. Groupby
print("\n6. Rata-rata nilai per kota:")
group_kota =
df_mahasiswa.groupby("kota")["nilai"].mean().reset_index()
print(group_kota)

# 7. DataFrame kedua untuk merge
df_jurusan = pd.DataFrame({
    "id": [1, 2, 3, 4, 5],
    "jurusan": ["TI", "SI", "TI", "MI", "SI"]
})

print("\n7. DataFrame jurusan:")
print(df_jurusan)

# 8. Merge
print("\n8. Merge mahasiswa + jurusan:")
df_gabung = pd.merge(df_mahasiswa, df_jurusan, on="id")
print(df_gabung)

# 9. Simpan ke CSV
df_gabung.to_csv("outputs/mahasiswa_gabung.csv", index=False)
print("\nFile berhasil disimpan:
outputs/mahasiswa_gabung.csv")

print("=" * 60)
print("Selesai. Anda sudah memakai Pandas untuk data nyata.")
print("=" * 60)
```

Jalankan dengan:

```
python src/pandas_latihan_terpadu.py
```

Program ini sangat baik untuk mengikat semua konsep inti dalam satu latihan praktis.

6.10 Langkah kerja yang disarankan untuk pembaca

Agar bab ini benar-benar hidup, berikut urutan praktik yang saya sarankan.

Tahap 1 — Mulai dari DataFrame kecil

- buat DataFrame manual dari dictionary
- tampilkan isi tabel
- pahami kolom dan baris

Tahap 2 — Masuk ke file nyata

- buat file CSV sederhana
- baca dengan `pd.read_csv()`
- cek `head()`, `shape`, `dtypes`, `info()`

Tahap 3 — Belajar memilih data

- ambil satu kolom
- ambil beberapa kolom
- filter data dengan kondisi tertentu
- urutkan data

Tahap 4 — Ringkas dan agregasi

- gunakan `groupby()`
- hitung sum, mean, min, max
- reset hasil agar menjadi DataFrame

Tahap 5 — Gabungkan beberapa sumber

- buat dua DataFrame
- gabungkan dengan `merge()`
- pahami hasil `inner`, `left`, dan `outer`

Tahap 6 — Simpan hasil

- ekspor ke CSV
- ekspor ke Excel jika perlu
- dokumentasikan apa yang ditemukan

Urutan ini akan membangun kebiasaan kerja yang sangat dekat dengan proyek data nyata.

6.11 Kesalahan umum pemula

Ada beberapa jebakan yang sering muncul saat pertama kali memakai Pandas.

1. Salah mengira Series dan DataFrame sama

   Padahal Series satu dimensi, DataFrame dua dimensi.

2. Lupa bahwa nama kolom sensitif

   `df["Nilai"]` berbeda dengan `df["nilai"]`.

3. Membaca kolom angka sebagai teks

   Karena itu `dtypes` dan `info()` penting diperiksa sejak awal.

4. Bingung antara loc dan iloc

   - `loc`: berdasarkan label
   - `iloc`: berdasarkan posisi

5. Salah memilih jenis merge

   Kalau tidak paham `inner`, `left`, `right`, `outer`, hasil gabungan bisa membingungkan.

6. Terlalu cepat lanjut tanpa ringkasan awal

   Padahal `head()`, `info()`, dan `describe()` adalah teman terbaik sebelum analisis lebih jauh.

6.12 Refleksi: dari tabel biasa ke alur kerja data

Bab ini mengajarkan sesuatu yang sangat penting: **data tabular bukan sekadar tabel untuk dilihat, tetapi bahan kerja untuk diprogram**. Dengan Pandas, pembaca mulai bergerak dari kebiasaan manual ala spreadsheet menuju alur kerja yang lebih kuat:

- membaca data
- memeriksa bentuk data
- memilih bagian penting
- memfilter
- mengelompokkan
- menggabungkan
- menyimpan hasil

Inilah inti mengapa DataFrame sering disebut sebagai **“Excel yang bisa diprogram”** dalam outline buku utama. Ia bukan menggantikan spreadsheet sepenuhnya, tetapi memperluas kemampuan pembaca untuk bekerja dengan data secara konsisten, cepat, dan dapat diulang.

Dalam konteks yang lebih besar, kemampuan di bab ini akan menjadi fondasi bagi:

- cleaning data
- statistik deskriptif
- visualisasi
- EDA
- pemilihan fitur untuk machine learning

Dengan kata lain, **siapa pun yang serius masuk ke AI/ML harus nyaman dulu dengan Pandas**.

Penutup Bab

Pandas adalah **alat utama untuk mengolah data nyata di Python**. Melalui bab ini, pembaca telah belajar memahami **Series dan DataFrame**, membaca **CSV dan Excel**, memilih kolom dan baris, melakukan **filtering** dan **sorting**, menghitung agregasi dengan `groupby()`, menggabungkan tabel dengan `merge()` dan `join()`, serta membuat **ringkasan data cepat**. Semua ini sangat sejalan dengan outline naskah utama dan referensi pendamping yang menempatkan Pandas sebagai pusat pekerjaan manipulasi data.

Pesan terpenting dari bab ini sederhana: **kalau NumPy membantu kita berpikir numerik, maka Pandas membantu kita berpikir tabular dan operasional**. Begitu pembaca nyaman memakai Pandas, langkah ke cleaning, visualisasi, statistik, dan machine learning akan terasa jauh lebih alami.

Tantangan kecil untuk pembaca: buat dua file CSV sederhana, satu berisi data mahasiswa dan satu lagi berisi data jurusan, lalu baca keduanya dengan Pandas, gabungkan dengan `merge()`, filter nilai di atas 80, dan hitung rata-rata nilai per jurusan. Begitu latihan ini berhasil, Anda sudah masuk ke pola kerja data yang sangat nyata.

Bab 7. Membersihkan Data Sebelum Analisis

Bab ini adalah **jantung dari pekerjaan data nyata**. Setelah pembaca mengenal NumPy dan Pandas, langkah berikutnya yang paling penting adalah memahami bahwa **data mentah hampir tidak pernah langsung siap dipakai**. Di outline naskah utama, bab ini memang diposisikan sebagai tahap sebelum analisis dan modeling, dengan fokus pada **missing value, duplicate, tipe data tidak sesuai, outlier dasar, encoding kategori, serta normalisasi dan standardisasi**. Outline tersebut juga menegaskan bahwa *cleaning* dan *preprocessing* adalah fondasi sebelum data masuk ke tahap statistik, visualisasi, atau model AI/ML.

Materi ini juga didukung kuat oleh dokumen referensi. Buku Jain menekankan bahwa dalam *feature engineering* dan *data preparation*, kita harus menangani **missing values**, melakukan **encoding** untuk variabel kategori, dan menerapkan **feature scaling** seperti *normalization* dan *standardization* agar model bekerja lebih baik. Referensi lain menunjukkan contoh konkret penggunaan `isnull()`, `dropna()`, `fillna()`, `duplicated()`, *one-hot encoding*, *z-score* untuk outlier, serta *scaling* dengan `MinMaxScaler`. Ada juga penekanan bahwa *data cleaning* bukan pekerjaan sampingan, tetapi proses yang memastikan integritas analisis dan kualitas model.

Bab ini ditulis agar **mudah dibaca oleh mahasiswa dan praktisi**, tetapi tetap terasa profesional dan aplikatif. Fokus utamanya adalah membuat pembaca mampu:

- **mendeteksi nilai hilang**
- **menangani data duplikat**
- **memperbaiki tipe data yang salah**
- **mengenali outlier dasar**
- **mengubah data kategori menjadi bentuk numerik**
- **memahami kapan perlu standarisasi dan normalisasi**

Semua contoh memakai **source code Python lengkap** dan dirancang untuk bisa dijalankan di **Ubuntu 26.04**.

7.1 Mengapa data harus dibersihkan sebelum analisis

Dalam praktik nyata, dataset sering datang dalam kondisi yang tidak ideal. Ada kolom yang kosong, baris yang terduplikasi, angka yang terbaca sebagai teks, nilai yang terlalu ekstrem, atau kategori yang belum bisa dipakai model. Kalau masalah ini diabaikan, hasil analisis bisa menyesatkan dan model bisa menjadi lemah atau bias. Salah satu referensi menegaskan bahwa **data cleaning and preprocessing** bukan sekadar langkah awal, tetapi proses vital yang mengubah data mentah menjadi fondasi yang layak untuk aplikasi AI.

Secara sederhana, kita bisa membayangkan proses ini seperti menyiapkan bahan masakan. Sebagus apa pun resepnya, kalau bahannya masih kotor, salah timbang, atau rusak, hasil akhirnya tidak akan bagus. Begitu juga dengan data. Karena itu, sebelum pembaca terlalu jauh membuat grafik atau melatih model, ada satu kebiasaan profesional yang harus dibangun: **cek dan rapikan datanya dulu**.

Masalah yang paling sering muncul dan akan kita bahas di bab ini adalah:

- **missing value**
- **data duplikat**
- **tipe data salah**
- **outlier dasar**
- **encoding kategori**
- **standarisasi dan normalisasi**

7.2 Dataset contoh yang akan dipakai

Agar pembahasan lebih hidup, kita akan memakai dataset sederhana bertema penjualan dan pelanggan. Dataset ini sengaja dibuat “sedikit berantakan” agar cocok untuk latihan *data cleaning*.

Simpan file berikut sebagai:

```
data/raw/data_kotor.csv
```

Isi file:

```
id,nama,kota,umur,pengeluaran,kategori_member
1,Andi,Jakarta,20,350000,Silver
2,Budi,Bandung,21,420000,Gold
3,Citra,Depok,,390000,Silver
4,Dina,Jakarta,22,1500000,Platinum
5,Eka,Bandung,20,410000,Gold
5,Eka,Bandung,20,410000,Gold
6,Fajar,Depok,dua puluh tiga,380000,Silver
7,Gina,Jakarta,24,,Gold
8,Hadi,Surabaya,23,365000,Silver
9,Ika,Jakarta,22,9999999,Platinum
10,Joko,,21,405000,Gold
```

Dari dataset ini, kita sengaja punya beberapa masalah:

- ada **umur kosong**
- ada **pengeluaran kosong**
- ada **data duplikat**
- ada **umur dengan teks “dua puluh tiga”**
- ada **kota kosong**
- ada **pengeluaran ekstrem** yang kemungkinan outlier
- ada **kolom kategori_member** yang bertipe kategorikal

Inilah bahan latihan yang sangat bagus untuk memahami pembersihan data.

7.3 Langkah awal: inspeksi dataset

Sebelum membersihkan apa pun, langkah pertama adalah **memahami kondisi awal data**. Beberapa referensi sangat menekankan pentingnya inspeksi awal menggunakan `head()`, `info()`, `describe()`, dan pemeriksaan nilai hilang.

Buat file:

```
src/inspeksi_data_kotor.py
```

Isi dengan:

```
import pandas as pd

df = pd.read_csv("data/raw/data_kotor.csv")

print("=== LIMA BARIS PERTAMA ===")
print(df.head())

print("\n=== SHAPE ===")
print(df.shape)

print("\n=== INFO ===")
print(df.info())

print("\n=== TIPE DATA ===")
print(df.dtypes)

print("\n=== RINGKASAN NUMERIK ===")
print(df.describe(include="all"))
```

Jalankan:

```
python src/inspeksi_data_kotor.py
```

Dari langkah ini, pembaca akan mulai melihat:

- berapa jumlah baris dan kolom
- kolom mana yang terbaca sebagai `object`
- kolom mana yang punya nilai kosong
- apakah kolom numerik sudah benar-benar numerik

Kebiasaan profesional: jangan pernah langsung menganalisis dataset tanpa inspeksi awal.

7.4 Missing value

Apa itu missing value

Missing value adalah nilai yang hilang atau kosong. Dalam Pandas, ini biasanya muncul sebagai `NaN`. Di data nyata, missing value bisa terjadi karena:

- form tidak diisi
- data gagal dikumpulkan
- integrasi sistem tidak lengkap
- kesalahan input
- data memang tidak tersedia

Referensi menekankan bahwa missing value yang dibiarkan begitu saja dapat mengurangi kualitas analisis dan melemahkan model. Strategi umum yang dipakai meliputi **menghapus baris**, **menghapus kolom**, atau **mengisi nilai hilang dengan mean/median/mode/interpolasi**.

Langkah 1 — Cek jumlah missing value

```
import pandas as pd

df = pd.read_csv("data/raw/data_kotor.csv")

print(df.isnull().sum())
```

Langkah 2 — Menghapus baris yang kosong

Kalau jumlah missing sangat sedikit dan aman untuk dihapus:

```
import pandas as pd

df = pd.read_csv("data/raw/data_kotor.csv")

df_drop = df.dropna()
print(df_drop)
```

Tetapi pendekatan ini sering terlalu keras, karena bisa membuang banyak data.

Langkah 3 — Mengisi missing value numerik

Untuk kolom numerik, pendekatan umum adalah memakai **mean** atau **median**. Jain menekankan bahwa mean, median, dan mode adalah strategi imputasi umum. Mean cocok untuk distribusi relatif normal, sedangkan median lebih aman untuk data yang *skewed* atau punya outlier.

Contoh:

```
import pandas as pd

df = pd.read_csv("data/raw/data_kotor.csv")

# Konversi umur dan pengeluaran ke numerik dulu jika perlu
df["umur"] = pd.to_numeric(df["umur"], errors="coerce")
df["pengeluaran"] = pd.to_numeric(df["pengeluaran"],
errors="coerce")

df["umur"] = df["umur"].fillna(df["umur"].median())
df["pengeluaran"] =
df["pengeluaran"].fillna(df["pengeluaran"].median())

print(df)
```

Langkah 4 — Mengisi missing value kategorikal

Untuk kolom seperti `kota` atau `kategori_member`, sering dipakai **mode** atau label khusus seperti `"Tidak Diketahui"`.

```
import pandas as pd

df = pd.read_csv("data/raw/data_kotor.csv")

df["kota"] = df["kota"].fillna(df["kota"].mode()[0])

print(df["kota"])
```

Langkah 5 — Alternatif dengan SimpleImputer

Kalau mulai masuk ke alur *machine learning*, pembaca bisa mengenal `SimpleImputer`. Jain memberi contoh penggunaan `SimpleImputer(strategy="mean")` untuk mengisi nilai hilang secara sistematis.

```
import pandas as pd
from sklearn.impute import SimpleImputer

df = pd.read_csv("data/raw/data_kotor.csv")

df["umur"] = pd.to_numeric(df["umur"], errors="coerce")
df["pengeluaran"] = pd.to_numeric(df["pengeluaran"],
errors="coerce")

imputer = SimpleImputer(strategy="median")
df[["umur", "pengeluaran"]] = imputer.fit_transform(df[["umur",
"pengeluaran"]])

print(df)
```

Kapan pakai dropna, kapan pakai fillna

Secara praktis:

- pakai `dropna()` jika data hilang sedikit dan tidak penting
- pakai `fillna()` jika kita ingin mempertahankan ukuran dataset
- pakai **median** untuk data numerik yang rawan outlier
- pakai **mode** untuk data kategori
- pakai **label khusus** bila ingin tetap menandai bahwa data memang hilang

7.5 Data duplikat

Apa itu data duplikat

Data duplikat berarti satu record muncul lebih dari sekali. Ini bisa menyebabkan:

- perhitungan total menjadi salah
- rata-rata bias
- jumlah observasi tampak lebih besar dari aslinya

Salah satu referensi mencontohkan validasi data dengan `duplicated().sum()` untuk memeriksa jumlah baris ganda.

Langkah 1 — Cek duplikat

```
import pandas as pd

df = pd.read_csv("data/raw/data_kotor.csv")

print("Jumlah duplikat:", df.duplicated().sum())
print(df[df.duplicated()])
```

Langkah 2 — Hapus duplikat

```
import pandas as pd

df = pd.read_csv("data/raw/data_kotor.csv")

df = df.drop_duplicates()

print("Jumlah baris setelah hapus duplikat:", len(df))
```

Langkah 3 — Duplikat berdasarkan kolom tertentu

Kadang kita tidak ingin membandingkan semua kolom, hanya kunci tertentu, misalnya `id`.

```
import pandas as pd

df = pd.read_csv("data/raw/data_kotor.csv")

duplikat_id = df[df.duplicated(subset=["id"],
keep=False)]
print(duplikat_id)
```

Praktik baik

- cek dulu sebelum hapus
- pahami apakah duplikat itu kesalahan atau memang transaksi yang sah
- kalau perlu, simpan versi asli sebelum `drop_duplicates()`

7.6 Tipe data salah

Mengapa tipe data penting

Salah satu masalah paling sering di data nyata adalah **angka terbaca sebagai teks**. Misalnya kolom umur berisi `"dua puluh tiga"` atau kolom harga terbaca sebagai `object` karena ada karakter asing. Kalau tipe data salah, maka:

- kita tidak bisa menghitung mean
- sorting bisa salah
- model machine learning bisa gagal

Langkah 1 — Cek tipe data

```
import pandas as pd

df = pd.read_csv("data/raw/data_kotor.csv")

print(df.dtypes)
```

Langkah 2 — Konversi ke numerik

Gunakan `pd.to_numeric(..., errors="coerce")`. Ini sangat berguna karena nilai yang tidak bisa dikonversi akan otomatis menjadi `NaN`, lalu bisa ditangani lagi sebagai missing value.

```
import pandas as pd

df = pd.read_csv("data/raw/data_kotor.csv")

df["umur"] = pd.to_numeric(df["umur"], errors="coerce")
df["pengeluaran"] = pd.to_numeric(df["pengeluaran"],
errors="coerce")

print(df.dtypes)
print(df)
```

Langkah 3 — Konversi tanggal bila perlu

Contoh bila ada kolom tanggal:

```
import pandas as pd

df["tanggal"] = pd.to_datetime(df["tanggal"], errors="coerce")
```

Praktik baik

- selalu cek `dtypes` setelah membaca file
- gunakan `to_numeric()` untuk angka
- gunakan `to_datetime()` untuk tanggal
- setelah konversi, periksa kembali nilai `NaN` yang muncul

7.7 Outlier dasar

Apa itu outlier

Outlier adalah nilai yang jauh berbeda dari mayoritas data. Referensi menekankan bahwa outlier dapat mendistorsi analisis dan prediksi, sehingga perlu dideteksi dan ditangani dengan hati-hati. Teknik yang umum dipakai adalah **Z-score** dan **IQR**.

Dalam dataset contoh, nilai `9999999` pada `pengeluaran` sangat mencurigakan. Bisa jadi itu benar, bisa jadi salah input. Intinya, outlier **tidak selalu harus dihapus**, tetapi setidaknya harus diperiksa.

A. Deteksi outlier dengan IQR

IQR (*Interquartile Range*) sering lebih aman untuk pemula dan cukup kuat terhadap distribusi yang tidak normal.

```
import pandas as pd

df = pd.read_csv("data/raw/data_kotor.csv")

df["pengeluaran"] = pd.to_numeric(df["pengeluaran"],
errors="coerce")

Q1 = df["pengeluaran"].quantile(0.25)
Q3 = df["pengeluaran"].quantile(0.75)
IQR = Q3 - Q1

batas_bawah = Q1 - 1.5 * IQR
batas_atas = Q3 + 1.5 * IQR

outlier = df[(df["pengeluaran"] < batas_bawah) |
(df["pengeluaran"] > batas_atas)]

print("Batas bawah:", batas_bawah)
print("Batas atas :", batas_atas)
print("\nData outlier:")
print(outlier)
```

B. Deteksi outlier dengan Z-score sederhana

Salah satu referensi memberi contoh *z-score* untuk mendeteksi outlier dengan ambang `> 3`.

```
import pandas as pd

df = pd.read_csv("data/raw/data_kotor.csv")

df["pengeluaran"] = pd.to_numeric(df["pengeluaran"],
errors="coerce")
mean_val = df["pengeluaran"].mean()
std_val = df["pengeluaran"].std()

df["z_score"] = (df["pengeluaran"] - mean_val) / std_val

outlier = df[df["z_score"].abs() > 3]
print(outlier)
```

C. Menangani outlier

Beberapa opsi umum:

- biarkan, jika memang valid
- hapus, jika jelas salah

- ganti dengan median
- *capping* ke batas tertentu

Contoh ganti dengan median:

```
import pandas as pd

df = pd.read_csv("data/raw/data_kotor.csv")

df["pengeluaran"] = pd.to_numeric(df["pengeluaran"],
errors="coerce")

Q1 = df["pengeluaran"].quantile(0.25)
Q3 = df["pengeluaran"].quantile(0.75)
IQR = Q3 - Q1

batas_bawah = Q1 - 1.5 * IQR
batas_atas = Q3 + 1.5 * IQR

median_pengeluaran = df["pengeluaran"].median()

df.loc[df["pengeluaran"] > batas_atas, "pengeluaran"] =
median_pengeluaran
df.loc[df["pengeluaran"] < batas_bawah, "pengeluaran"] =
median_pengeluaran

print(df)
```

Praktik baik

- jangan otomatis hapus outlier tanpa konteks
- cek dulu apakah itu kesalahan input
- gunakan IQR untuk pendekatan yang mudah dan stabil
- dokumentasikan keputusan Anda

7.8 Encoding kategori

Mengapa perlu encoding

Sebagian besar algoritma machine learning membutuhkan input numerik. Karena itu, kolom kategori seperti `"Silver"`, `"Gold"`, atau `"Platinum"` harus diubah ke bentuk numerik. Jain menegaskan bahwa *categorical variables must be encoded* sebelum masuk ke model, dengan teknik umum berupa **One-Hot Encoding** dan **Label Encoding**.

A. One-hot encoding dengan Pandas

Cara paling mudah untuk pemula adalah `pd.get_dummies()`.

```
import pandas as pd

df = pd.read_csv("data/raw/data_kotor.csv")

encoded = pd.get_dummies(df, columns=["kategori_member"], dtype=int)

print(encoded.head())
```

Hasilnya kira-kira akan menambah kolom seperti:

- `kategori_member_Gold`
- `kategori_member_Platinum`
- `kategori_member_Silver`

B. One-hot encoding dengan scikit-learn

Jain juga memberi contoh `OneHotEncoder` dari scikit-learn.

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

df = pd.DataFrame({
    "warna": ["Merah", "Biru", "Hijau"]
})

encoder = OneHotEncoder(sparse_output=False)
hasil = encoder.fit_transform(df[["warna"]])

print(hasil)
```

C. Label encoding sederhana

Untuk kategori yang memang berurutan, misalnya `"Rendah"`, `"Sedang"`, `"Tinggi"`, kadang kita bisa memetakan manual.

```
import pandas as pd

df = pd.DataFrame({
    "prioritas": ["Rendah", "Sedang", "Tinggi", "Sedang"]
})

mapping = {
    "Rendah": 1,
    "Sedang": 2,
    "Tinggi": 3
}

df["prioritas_encoded"] = df["prioritas"].map(mapping)
print(df)
```

Kapan pakai one-hot, kapan label encoding

- pakai **one-hot encoding** untuk kategori nominal tanpa urutan alami
- pakai **label encoding** bila memang ada urutan logis

Referensi lain juga menunjukkan penggunaan **label encoding** pada kategori tertentu untuk kebutuhan model klasifikasi.

7.9 Standarisasi dan normalisasi

Mengapa perlu scaling

Jain menegaskan bahwa *feature scaling* membantu memastikan semua fitur berkontribusi lebih seimbang terhadap model. Dua teknik yang paling umum adalah:

- **Normalization**: mengubah rentang ke 0–1
- **Standardization**: membuat mean = 0 dan std = 1

Ini penting terutama untuk model yang sensitif terhadap skala, seperti:

- KNN
- SVM
- K-Means
- regresi dengan regularisasi

A. Normalisasi dengan MinMaxScaler

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

df = pd.read_csv("data/raw/data_kotor.csv")

df["umur"] = pd.to_numeric(df["umur"], errors="coerce")
df["pengeluaran"] = pd.to_numeric(df["pengeluaran"], errors="coerce")

df["umur"] = df["umur"].fillna(df["umur"].median())
df["pengeluaran"] =
df["pengeluaran"].fillna(df["pengeluaran"].median())

scaler = MinMaxScaler()
df[["umur_normal", "pengeluaran_normal"]] =
scaler.fit_transform(df[["umur", "pengeluaran"]])

print(df[["umur", "umur_normal", "pengeluaran",
"pengeluaran_normal"]])
```

Referensi Lim juga menunjukkan penggunaan `MinMaxScaler` untuk menskalakan data ke rentang 0–1.

B. Standardisasi dengan StandardScaler

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

df = pd.read_csv("data/raw/data_kotor.csv")

df["umur"] = pd.to_numeric(df["umur"], errors="coerce")
df["pengeluaran"] = pd.to_numeric(df["pengeluaran"],
errors="coerce")

df["umur"] = df["umur"].fillna(df["umur"].median())
df["pengeluaran"] =
df["pengeluaran"].fillna(df["pengeluaran"].median())

scaler = StandardScaler()
df[["umur_std", "pengeluaran_std"]] =
scaler.fit_transform(df[["umur", "pengeluaran"]])

print(df[["umur", "umur_std", "pengeluaran",
"pengeluaran_std"]])
```

Kapan pakai normalisasi, kapan standardisasi

Secara praktis:

- **normalisasi** cocok kalau ingin rentang tetap 0–1
- **standardisasi** cocok kalau ingin data terpusat di sekitar nol dan sebaran distandarkan
- untuk pemula, pahami dulu bahwa keduanya adalah cara membuat fitur berbeda skala menjadi lebih seimbang

7.10 Source code Python lengkap: membersihkan data langkah demi langkah

Berikut **program lengkap** yang menggabungkan seluruh tahap pada bab ini.

Simpan sebagai:

```
src/cleaning_data_lengkap.py
```

Isi dengan:

```
import pandas as pd
from sklearn.preprocessing import MinMaxScaler,
StandardScaler

print("=" * 70)
print("MEMBERSIHKAN DATA SEBELUM ANALISIS")
print("=" * 70)

# 1. Baca data
df = pd.read_csv("data/raw/data_kotor.csv")

print("\n[1] DATA AWAL")
print(df)
print("\nShape awal:", df.shape)

# 2. Cek missing value
print("\n[2] JUMLAH MISSING VALUE")
print(df.isnull().sum())

# 3. Hapus duplikat
df = df.drop_duplicates()
print("\n[3] SETELAH HAPUS DUPLIKAT")
print("Shape sekarang:", df.shape)

# 4. Perbaiki tipe data
df["umur"] = pd.to_numeric(df["umur"], errors="coerce")
df["pengeluaran"] = pd.to_numeric(df["pengeluaran"],
errors="coerce")

print("\n[4] TIPE DATA SETELAH KONVERSI")
print(df.dtypes)

# 5. Isi missing value
df["umur"] = df["umur"].fillna(df["umur"].median())
df["pengeluaran"] =
df["pengeluaran"].fillna(df["pengeluaran"].median())
df["kota"] = df["kota"].fillna(df["kota"].mode()[0])
```

```
print("\n[5] SETELAH MENANGANI MISSING VALUE")
print(df)

# 6. Deteksi outlier dengan IQR pada kolom pengeluaran
Q1 = df["pengeluaran"].quantile(0.25)
Q3 = df["pengeluaran"].quantile(0.75)
IQR = Q3 - Q1
batas_bawah = Q1 - 1.5 * IQR
batas_atas = Q3 + 1.5 * IQR

print("\n[6] BATAS OUTLIER")
print("Batas bawah:", batas_bawah)
print("Batas atas :", batas_atas)

outlier = df[(df["pengeluaran"] < batas_bawah) |
(df["pengeluaran"] > batas_atas)]
print("\nData outlier:")
print(outlier)

# Tangani outlier dengan median
median_pengeluaran = df["pengeluaran"].median()
df.loc[df["pengeluaran"] > batas_atas, "pengeluaran"] =
median_pengeluaran
df.loc[df["pengeluaran"] < batas_bawah, "pengeluaran"] =
median_pengeluaran

print("\n[7] SETELAH MENANGANI OUTLIER")
print(df)

# 7. Encoding kategori
df_encoded = pd.get_dummies(df, columns=["kategori_member"],
dtype=int)

print("\n[8] SETELAH ONE-HOT ENCODING")
print(df_encoded.head())

# 8. Normalisasi
scaler_minmax = MinMaxScaler()
df_encoded[["umur_normal", "pengeluaran_normal"]] =
scaler_minmax.fit_transform(
    df_encoded[["umur", "pengeluaran"]]
)

# 9. Standardisasi
scaler_std = StandardScaler()
df_encoded[["umur_std", "pengeluaran_std"]] =
scaler_std.fit_transform(
    df_encoded[["umur", "pengeluaran"]]
```

```
)

print("\n[9] SETELAH NORMALISASI DAN STANDARDISASI")
print(df_encoded[[
    "umur", "umur_normal", "umur_std",
    "pengeluaran", "pengeluaran_normal", "pengeluaran_std"
]])

# 10. Simpan hasil
df_encoded.to_csv("outputs/data_bersih.csv", index=False)

print("\n[10] FILE BERHASIL DISIMPAN")
print("outputs/data_bersih.csv")

print("\nSELESAI. DATA SIAP DIPAKAI UNTUK ANALISIS
LANJUTAN.")
print("=" * 70)
```

Jalankan dengan:

```
python src/cleaning_data_lengkap.py
```

7.11 Urutan kerja yang disarankan

Agar pembaca membangun kebiasaan yang benar, berikut urutan kerja yang saya sarankan setiap kali menerima dataset baru:

- **Langkah 1: baca data**
- **Langkah 2: cek struktur** dengan `head()`, `info()`, `dtypes`, `describe()`
- **Langkah 3: cek missing value**
- **Langkah 4: cek duplikat**
- **Langkah 5: pastikan tipe data benar**
- **Langkah 6: identifikasi outlier**
- **Langkah 7: encoding kategori bila perlu**
- **Langkah 8: scaling jika akan masuk model**
- **Langkah 9: simpan versi data bersih**
- **Langkah 10: dokumentasikan keputusan pembersihan**

Urutan ini sangat selaras dengan penekanan referensi bahwa *data cleaning and preprocessing* adalah fondasi sebelum analisis dan AI

7.12 Kesalahan umum pemula

Ada beberapa jebakan yang sering terjadi.

- **Langsung `dropna()` semua** tanpa memikirkan dampak ke ukuran data.
- **Menghapus outlier otomatis** padahal bisa jadi nilai itu valid.
- **Lupa cek `dtypes`**, sehingga angka masih terbaca sebagai teks.
- **Encoding kategori sembarangan**, misalnya memberi angka 1, 2, 3 untuk kategori yang sebenarnya tidak berurutan.
- **Scaling sebelum data dibersihkan**, padahal nilai hilang dan outlier bisa merusak hasil scaling.
- **Tidak menyimpan versi data bersih**, sehingga sulit mengulang eksperimen.

Kebiasaan yang baik adalah selalu bertanya:
apakah keputusan pembersihan ini masuk akal secara data dan konteks?

7.13 Refleksi: membersihkan data berarti membangun kepercayaan pada analisis

Bab ini sebetulnya bukan hanya tentang perintah Pandas atau scikit-learn. Yang lebih penting adalah perubahan cara berpikir. Pembaca mulai belajar bahwa sebelum kita percaya pada grafik, statistik, atau model, kita harus lebih dulu percaya pada kualitas data yang kita pakai. Salah satu referensi menyebut bahwa dokumentasi dan transparansi dalam preprocessing sangat penting untuk validitas keputusan dan akuntabilitas proyek AI.

Dengan kata lain, *data cleaning* adalah proses membangun **kepercayaan**:

- percaya bahwa nilai kosong sudah ditangani dengan benar
- percaya bahwa duplikat tidak menyesatkan hasil
- percaya bahwa tipe data sesuai
- percaya bahwa outlier tidak merusak kesimpulan
- percaya bahwa fitur numerik siap dipakai model

Inilah kenapa bab ini sangat penting. Tanpa fondasi ini, analisis yang terlihat canggih bisa berdiri di atas data yang rapuh.

Penutup Bab

Membersihkan data sebelum analisis adalah **keterampilan inti** untuk setiap calon praktisi data, AI, dan machine learning. Melalui bab ini, pembaca telah belajar menangani **missing value**, **data duplikat**, **tipe data salah**, **outlier dasar**, **encoding kategori**, serta **standardisasi dan normalisasi**. Semua topik ini memang sudah ditegaskan dalam outline naskah utama sebagai fondasi sebelum modeling, dan sangat didukung oleh materi lampiran yang menekankan imputasi, encoding, dan scaling sebagai bagian penting dari *data preparation*.

Pesan terpenting dari bab ini sederhana: **model yang baik tidak bisa menyelamatkan data yang buruk**. Karena itu, semakin cepat pembaca nyaman dengan proses pembersihan data, semakin kuat fondasi untuk bab-bab berikutnya, seperti visualisasi, statistik, EDA, dan machine learning.

Tantangan kecil untuk pembaca: modifikasi program `cleaning_data_lengkap.py` agar selain menyimpan `data_bersih.csv`, program juga membuat satu kolom baru bernama `segment_pengeluaran` yang membagi pelanggan menjadi tiga kategori: `Rendah`, `Sedang`, dan `Tinggi` berdasarkan nilai pengeluaran.

Bab 8. Visualisasi Data yang Membantu Berpikir

Bab ini sangat penting karena pada titik inilah pembaca mulai bergerak dari sekadar **mengolah tabel** menjadi **melihat pola, tren, sebaran, dan hubungan antar-variabel**. Di outline naskah utama, bab visualisasi memang diletakkan setelah Pandas dan *data cleaning*, dengan fokus pada **prinsip visualisasi yang efektif**, **line plot**, **bar plot**, **scatter plot**, **histogram**, **boxplot**, **heatmap korelasi**, serta pembahasan tentang **kapan grafik justru menyesatkan** dan pengantar **storytelling dengan visual**. Struktur ini sangat tepat, karena visualisasi seharusnya menjadi jembatan antara data mentah dan pemahaman manusia.

Materi dari dokumen lampiran juga mendukung pendekatan ini. Jain menjelaskan *use case* dasar untuk **bar chart**, **scatter plot**, **histogram**, dan **heatmap**, serta memberi contoh bahwa **line plot** sangat cocok untuk melihat tren deret waktu, sedangkan **heatmap korelasi** membantu memvisualisasikan hubungan antar-variabel. Referensi lain menegaskan bahwa **histogram** berguna untuk melihat distribusi, **boxplot** sangat baik untuk melihat *outlier* dan sebaran, dan **heatmap** umum dipakai untuk matriks korelasi. Saltz juga menekankan bahwa *boxplot* membantu melihat *skewness* dan *outlier* dengan cepat, serta bahwa pemilihan jumlah *bins* pada histogram memengaruhi cara kita membaca distribusi.

Bab ini disusun agar **mudah dibaca oleh mahasiswa dan praktisi**, tetapi tetap profesional dan aplikatif. Fokusnya adalah membuat pembaca mampu:

- **memilih jenis grafik yang sesuai dengan pertanyaan analisis**
- **membuat grafik dasar dengan Python**
- **membaca grafik dengan benar**
- **menghindari jebakan interpretasi**
- **menggunakan visualisasi untuk membantu berpikir, bukan sekadar mempercantik laporan**

Semua contoh memakai **source code Python lengkap** dan dirancang untuk berjalan di **Ubuntu 26.04**.

8.1 Mengapa visualisasi data penting

Data sering kali terlalu banyak atau terlalu datar bila hanya dilihat sebagai angka dalam tabel. Visualisasi membantu mata dan otak manusia menangkap pola lebih cepat. Misalnya:

- apakah penjualan naik dari bulan ke bulan
- kategori mana yang paling tinggi
- distribusi nilai mahasiswa cenderung rapat atau menyebar
- apakah ada *outlier*
- apakah dua variabel tampak berhubungan
- apakah beberapa fitur saling berkorelasi kuat

Jain menekankan bahwa visualisasi data adalah alat penting untuk **mengkomunikasikan insight dan menceritakan kisah dengan data**, sedangkan outline utama buku ini menempatkannya sebagai bagian dari **EDA yang membantu berpikir**.

Prinsip yang perlu dipegang pembaca sejak awal adalah:

- **grafik bukan pajangan**
- **grafik adalah alat berpikir**
- **grafik yang bagus membantu keputusan**
- **grafik yang buruk bisa menyesatkan**

Karena itu, saat membuat grafik, selalu mulai dari pertanyaan:

> **Saya ingin menjawab apa dari data ini?**

Bila pertanyaan itu jelas, pemilihan grafik biasanya akan lebih tepat.

8.2 Menyiapkan *library* visualisasi di Ubuntu 26.04

Untuk bab ini, kita akan memakai tiga *library* utama:

- **Pandas** untuk membaca dan menyiapkan data
- **Matplotlib** untuk membuat grafik dasar
- **Seaborn** untuk grafik statistik yang lebih ringkas dan rapi

Pasang jika belum ada:

```
pip install pandas matplotlib seaborn
```

Lalu buat file `src/cek_visualisasi.py`:

```
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns

print("Pandas OK :", pd.__version__)
print("Seaborn OK:", sns.__version__)
print("Matplotlib OK:", plt.matplotlib.__version__)
```

Jalankan:

```
python src/cek_visualisasi.py
```

Kalau berhasil, pembaca siap mulai membuat grafik.

8.3 Dataset contoh untuk bab ini

Agar pembahasan lebih hidup, kita akan memakai dataset kecil bertema penjualan bulanan.

Simpan file berikut sebagai:

```
data/raw/penjualan_visual.csv
```

Isi file:

```
bulan,penjualan,biaya,produk_terjual,kategori,iklan
Jan,120,80,30,A,10
Feb,135,90,34,A,12
Mar,128,85,31,B,11
Apr,150,95,36,B,14
Mei,170,110,40,A,16
Jun,165,108,39,C,15
Jul,180,120,43,C,18
Agu,175,118,42,B,17
Sep,160,105,38,A,14
Okt,190,125,45,C,19
Nov,200,130,47,B,20
Des,210,138,50,A,22
```

Dataset ini cukup baik untuk banyak jenis visualisasi karena punya:

- urutan waktu (`bulan`)
- nilai numerik (`penjualan`, `biaya`, `produk_terjual`, `iklan`)
- kategori (`kategori`)

8.4 Prinsip visualisasi yang efektif

Sebelum masuk ke jenis grafik satu per satu, ada beberapa prinsip penting yang sejalan dengan semangat outline utama: **visualisasi harus membantu berpikir, bukan membingungkan**.

Prinsip dasar yang perlu dipegang

- **Pilih grafik sesuai pertanyaan**
  - tren waktu → *line plot*
  - perbandingan kategori → *bar plot*
  - distribusi → *histogram*
  - *outlier* dan sebaran → *boxplot*
  - hubungan dua variabel → *scatter plot*
  - hubungan banyak variabel → *heatmap korelasi*

- **Beri judul yang jelas**
  Judul harus menjelaskan isi grafik, bukan hanya menulis “Grafik 1”.
- **Label sumbu harus lengkap**
  Tanpa label sumbu, pembaca mudah salah tafsir.
- **Warna secukupnya**
  Jain mengingatkan bahwa terlalu banyak warna justru membuat grafik membingungkan.
- **Jangan memanipulasi tampilan secara berlebihan**
  Skala yang tidak proporsional, *bins* yang aneh, atau pemotongan sumbu yang tidak dijelaskan bisa menyesatkan pembaca.
- **Fokus pada cerita yang ingin disampaikan**
  Grafik yang baik selalu menjawab pertanyaan analitis tertentu.

8.5 *Line plot*

Kapan memakai *line plot*

Line plot paling cocok untuk **data berurutan**, terutama deret waktu. Jain menjelaskan bahwa untuk *time series*, *line plot* membantu mengidentifikasi tren dari waktu ke waktu.

Pertanyaan yang cocok dijawab dengan *line plot*:

- apakah penjualan naik dari bulan ke bulan
- kapan terjadi lonjakan
- apakah ada pola musiman sederhana
- apakah dua metrik bergerak bersama

Source code lengkap

Buat file `src/line_plot_penjualan.py`:

```
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns

# Baca data
df = pd.read_csv("data/raw/penjualan_visual.csv")

# Buat line plot
plt.figure(figsize=(10, 5))
sns.lineplot(data=df, x="bulan", y="penjualan", marker="o")

plt.title("Tren Penjualan Bulanan")
plt.xlabel("Bulan")
plt.ylabel("Penjualan")
plt.grid(True)
plt.tight_layout()
plt.show()
```

Apa yang perlu dibaca dari grafik ini

Saat melihat *line plot*, perhatikan:

- arah umum garis: naik, turun, atau datar
- titik lonjakan atau penurunan tajam
- pola berulang
- kestabilan atau volatilitas data

Interpretasi contoh

> Dari dataset contoh, pembaca kemungkinan akan melihat tren penjualan yang cenderung naik dari awal tahun ke akhir tahun. Ini memberi sinyal bahwa bisnis berkembang, atau ada musim penjualan yang lebih kuat di akhir tahun.

Kesalahan umum

- memakai *line plot* untuk kategori yang tidak berurutan
- terlalu banyak garis hingga grafik sulit dibaca
- tidak memberi penanda atau label yang cukup

8.6 *Bar plot*

Kapan memakai *bar plot*

> Menurut Jain, *bar chart* cocok untuk **membandingkan kategori atau grup**. Contohnya penjualan per wilayah, jumlah mahasiswa per jurusan, atau omzet per kategori produk.

Source code lengkap

> Buat file `src/bar_plot_kategori.py`:

```
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns

df = pd.read_csv("data/raw/penjualan_visual.csv")

# Ringkas total penjualan per kategori
df_kategori = df.groupby("kategori")["penjualan"].sum().reset_index()

plt.figure(figsize=(8, 5))
sns.barplot(data=df_kategori, x="kategori", y="penjualan")

plt.title("Total Penjualan per Kategori")
plt.xlabel("Kategori")
plt.ylabel("Total Penjualan")
plt.tight_layout()
plt.show()
```

Apa yang perlu dibaca dari grafik ini

Perhatikan:

- kategori mana yang paling tinggi
- kategori mana yang paling rendah
- apakah selisih antar kategori besar atau kecil

Interpretasi contoh

Kalau kategori A lebih tinggi dari B dan C, maka kategori A memberi kontribusi terbesar terhadap total penjualan pada dataset ini.

Kesalahan umum

- terlalu banyak kategori sehingga label sulit dibaca
- urutan kategori tidak masuk akal
- memakai *bar plot* untuk data yang sebetulnya lebih cocok dilihat sebagai tren waktu

8.7 Histogram

Kapan memakai histogram

Jain menjelaskan bahwa histogram dipakai untuk melihat **distribusi satu variabel**. Saltz juga menekankan bahwa histogram membagi data ke dalam *bins* dan memperlihatkan frekuensi kemunculan nilai dalam tiap rentang, sehingga membantu memahami bentuk distribusi.

Histogram cocok untuk pertanyaan seperti:

- apakah nilai terkonsentrasi di tengah
- apakah distribusi miring ke kanan atau kiri
- apakah ada kemungkinan *outlier*
- apakah data tampak simetris atau tidak

Source code lengkap

Buat file `src/histogram_penjualan.py`:

```
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns

df = pd.read_csv("data/raw/penjualan_visual.csv")

plt.figure(figsize=(8, 5))
sns.histplot(df["penjualan"], bins=6, kde=False)

plt.title("Histogram Penjualan")
plt.xlabel("Penjualan")
plt.ylabel("Frekuensi")
plt.tight_layout()
plt.show()
```

Tentang *bins*

Saltz mengingatkan bahwa jumlah *bins* mempengaruhi tampilan distribusi. Terlalu sedikit bisa menyembunyikan detail, terlalu banyak bisa membuat pola sulit dibaca.

Apa yang perlu dibaca

Perhatikan:

- pusat distribusi
- lebar sebaran
- kemiringan (*skewness*)
- kemungkinan ada ekor panjang
- kemungkinan nilai ekstrim

Interpretasi contoh

Kalau sebagian besar nilai penjualan terkumpul di rentang tinggi, artinya performa bisnis lebih banyak berada pada bulan dengan penjualan besar.

Variasi dengan KDE

Jain juga menunjukkan bahwa histogram bisa dikombinasikan dengan KDE untuk memberi kurva halus distribusi.

```
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns

df = pd.read_csv("data/raw/penjualan_visual.csv")

plt.figure(figsize=(8, 5))
sns.histplot(df["penjualan"], bins=6, kde=True)

plt.title("Histogram dan KDE Penjualan")
plt.xlabel("Penjualan")
plt.ylabel("Frekuensi / Kepadatan")
plt.tight_layout()
plt.show()
```

8.8 Boxplot

Kapan memakai boxplot

Boxplot sangat berguna untuk melihat:

- sebaran data
- median
- kuartil
- *outlier*

Jain menjelaskan bahwa boxplot menampilkan ringkasan lima angka: minimum, kuartil 1, median, kuartil 3, dan maksimum. Saltz menambahkan bahwa boxplot memungkinkan analis melihat *skewness* dan *outlier* secara cepat, dan titik yang jauh dari kotak sering ditandai sebagai outlier berdasarkan aturan 1,5 × IQR.

Source code lengkap

Buat file `src/boxplot_penjualan.py`:

```
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns

df = pd.read_csv("data/raw/penjualan_visual.csv")

plt.figure(figsize=(8, 5))
sns.boxplot(y=df["penjualan"])

plt.title("Boxplot Penjualan")
plt.ylabel("Penjualan")
plt.tight_layout()
plt.show()
```

Boxplot per kategori

```
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns

df = pd.read_csv("data/raw/penjualan_visual.csv")

plt.figure(figsize=(8, 5))
sns.boxplot(data=df, x="kategori", y="penjualan")

plt.title("Boxplot Penjualan per Kategori")
plt.xlabel("Kategori")
plt.ylabel("Penjualan")
plt.tight_layout()
plt.show()
```

Apa yang perlu dibaca

Perhatikan:

- posisi median
- tinggi kotak, yang mencerminkan IQR
- panjang whisker
- titik terpisah yang mungkin outlier
- perbedaan distribusi antar kategori

Interpretasi contoh

Kalau kategori tertentu memiliki kotak lebih tinggi, berarti variasi penjualannya lebih besar. Kalau ada titik jauh di luar whisker, itu sinyal *outlier* yang perlu diperiksa.

8.9 Scatter plot

Kapan memakai scatter plot

Jain menjelaskan bahwa *scatter plot* cocok untuk melihat **hubungan dua variabel numerik**. Ini sangat penting saat pembaca ingin tahu apakah dua kolom bergerak bersama.

Contoh pertanyaan:

- apakah iklan lebih tinggi terkait dengan penjualan yang lebih besar
- apakah biaya naik saat penjualan naik
- apakah dua variabel tampak memiliki hubungan linier

Source code lengkap

Buat file `src/scatter_plot_penjualan.py`:

```
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns

df = pd.read_csv("data/raw/penjualan_visual.csv")

plt.figure(figsize=(8, 5))
sns.scatterplot(data=df, x="iklan", y="penjualan",
hue="kategori", s=100)

plt.title("Hubungan Iklan dan Penjualan")
plt.xlabel("Biaya Iklan")
plt.ylabel("Penjualan")
plt.tight_layout()
plt.show()
```

Apa yang perlu dibaca

Perhatikan:

- apakah titik-titik cenderung naik bersama
- apakah ada pola linier
- apakah ada kelompok terpisah
- apakah ada titik ekstrim

Interpretasi contoh

Kalau titik-titik membentuk pola menaik, itu memberi sinyal bahwa biaya iklan yang lebih tinggi mungkin terkait dengan penjualan yang lebih tinggi. Ini belum membuktikan sebab-akibat, tetapi cukup untuk memberi petunjuk awal.

Catatan penting

Scatter plot menunjukkan **hubungan**, bukan otomatis **kausalitas**. Ini penting agar pembaca tidak terlalu cepat menarik kesimpulan.

8.10 Heatmap korelasi

Kapan memakai heatmap korelasi

Jain menegaskan bahwa **correlation matrix heatmap** adalah cara yang berguna untuk memvisualisasikan hubungan antara banyak variabel sekaligus. Referensi lain juga menyebut heatmap sebagai alat umum untuk matriks korelasi maupun matriks lain yang nilainya ingin dilihat melalui intensitas warna.

Source code lengkap

Buat file `src/heatmap_korelasi.py`:

```
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns

df = pd.read_csv("data/raw/penjualan_visual.csv")

# Ambil hanya kolom numerik
df_numerik = df[["penjualan", "biaya", "produk_terjual",
"iklan"]]

# Hitung korelasi
corr_matrix = df_numerik.corr()

plt.figure(figsize=(8, 6))
sns.heatmap(corr_matrix, annot=True, cmap="coolwarm")

plt.title("Heatmap Korelasi")
plt.tight_layout()
plt.show()
```

Apa yang perlu dibaca

Perhatikan:

- nilai korelasi mendekati **1** → hubungan positif kuat
- nilai korelasi mendekati **-1** → hubungan negatif kuat
- nilai mendekati **0** → hubungan linier lemah

Interpretasi contoh

Kalau `iklan` dan `penjualan` punya korelasi tinggi, itu berarti saat iklan naik, penjualan juga cenderung naik. Kalau `biaya` dan `penjualan` juga tinggi korelasinya, mungkin karena biaya operasional meningkat saat aktivitas bisnis naik.

Batasan

Korelasi tidak sama dengan sebab-akibat. Heatmap sangat berguna untuk eksplorasi awal, tetapi bukan bukti final.

8.11 Cara membaca grafik dengan benar

Ini bagian yang paling penting. Outline utama buku menekankan bukan hanya jenis grafik, tetapi juga **kapan grafik justru menyesatkan**.

A. Selalu mulai dari pertanyaan

Jangan lihat grafik hanya sebagai gambar. Tanyakan:

- pertanyaan apa yang ingin dijawab
- variabel apa yang dibandingkan
- apakah grafik ini memang bentuk yang paling tepat

B. Baca judul, sumbu, dan satuan

Banyak kesalahan interpretasi terjadi hanya karena:

- sumbu X tidak jelas
- sumbu Y tidak punya satuan
- judul tidak menjelaskan konteks

C. Perhatikan skala

Skala sangat mempengaruhi persepsi. Misalnya:

- sumbu Y yang dipotong bisa membuat perubahan kecil tampak dramatis
- *bins* histogram yang berbeda bisa membuat distribusi terlihat berbeda
- skala warna heatmap yang buruk bisa menyamarkan perbedaan penting

Saltz menekankan bahwa pemilihan *bins* dan tampilan visual memengaruhi bagaimana distribusi dipersepsikan.

D. Bedakan distribusi, tren, dan hubungan

- *histogram* → distribusi
- *line plot* → tren
- *scatter plot* → hubungan
- *boxplot* → sebaran dan outlier
- *bar plot* → perbandingan kategori

Kalau salah memilih grafik, pembaca bisa salah membaca data.

E. Jangan terlalu cepat menyimpulkan kausalitas

Kalau dua variabel berkorelasi, itu belum berarti yang satu menyebabkan yang lain. Ini terutama penting saat membaca scatter plot dan heatmap korelasi.

F. Perhatikan outlier

Kadang satu titik ekstrim bisa mengubah bentuk grafik atau korelasi secara besar. Karena itu, jika ada nilai yang sangat jauh, periksa dulu apakah itu kesalahan input atau memang kasus spesial.

G. Cari cerita, bukan sekadar angka

Grafik yang baik membantu kita menjawab:

- apa yang sedang terjadi
- apa yang tidak biasa
- apa yang layak ditindaklanjuti

Di sinilah visualisasi berubah menjadi alat berpikir, bukan hanya hiasan.

8.12 Source code Python lengkap: latihan terpadu visualisasi

Berikut satu program lengkap yang memadukan semua grafik utama dalam bab ini.

Simpan sebagai `src/visualisasi_terpadu.py`:

```
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns

# Baca data
df = pd.read_csv("data/raw/penjualan_visual.csv")

# Atur gaya visual sederhana
sns.set_theme(style="whitegrid")

print("=== DATA AWAL ===")
print(df.head())

# 1. Line plot
plt.figure(figsize=(10, 5))
sns.lineplot(data=df, x="bulan", y="penjualan", marker="o")
plt.title("Tren Penjualan Bulanan")
plt.xlabel("Bulan")
plt.ylabel("Penjualan")
plt.tight_layout()
plt.show()

# 2. Bar plot
df_kategori = df.groupby("kategori")["penjualan"].sum().reset_index()
```

```
plt.figure(figsize=(8, 5))
sns.barplot(data=df_kategori, x="kategori", y="penjualan")
plt.title("Total Penjualan per Kategori")
plt.xlabel("Kategori")
plt.ylabel("Total Penjualan")
plt.tight_layout()
plt.show()

# 3. Histogram
plt.figure(figsize=(8, 5))
sns.histplot(df["penjualan"], bins=6, kde=True)
plt.title("Histogram Penjualan")
plt.xlabel("Penjualan")
plt.ylabel("Frekuensi")
plt.tight_layout()
plt.show()

# 4. Boxplot
plt.figure(figsize=(8, 5))
sns.boxplot(data=df, x="kategori", y="penjualan")
plt.title("Boxplot Penjualan per Kategori")
plt.xlabel("Kategori")
plt.ylabel("Penjualan")
plt.tight_layout()
plt.show()

# 5. Scatter plot
plt.figure(figsize=(8, 5))
sns.scatterplot(data=df, x="iklan", y="penjualan", hue="kategori", s=100)
plt.title("Hubungan Iklan dan Penjualan")
plt.xlabel("Biaya Iklan")
plt.ylabel("Penjualan")
plt.tight_layout()
plt.show()

# 6. Heatmap korelasi
df_numerik = df[["penjualan", "biaya", "produk_terjual", "iklan"]]
corr_matrix = df_numerik.corr()

plt.figure(figsize=(8, 6))
sns.heatmap(corr_matrix, annot=True, cmap="coolwarm")
plt.title("Heatmap Korelasi")
plt.tight_layout()
plt.show()

print("Selesai. Anda sudah membuat visualisasi inti untuk EDA dasar.")
```

Jalankan dengan:

```
python src/visualisasi_terpadu.py
```

Program ini sangat baik sebagai latihan terpadu sebelum pembaca masuk ke bab statistik praktis dan mini project EDA.

8.13 Langkah kerja yang disarankan untuk pembaca

Agar bab ini benar-benar melatih kebiasaan analitis, berikut urutan praktik yang disarankan.

- **Langkah 1: baca data**
  Gunakan Pandas untuk memuat dataset.
- **Langkah 2: tentukan pertanyaan**
  Misalnya ingin melihat tren, membandingkan kategori, atau memeriksa hubungan dua variabel.
- **Langkah 3: pilih grafik yang sesuai**
  Jangan mulai dari grafik; mulai dari pertanyaannya.
- **Langkah 4: buat grafik sederhana**
  Tambahkan judul, label sumbu, dan ukuran yang nyaman dibaca.
- **Langkah 5: interpretasikan**
  Tuliskan 2–3 kalimat tentang apa yang terlihat.
- **Langkah 6: cek apakah ada jebakan**
  Misalnya outlier, skala aneh, atau korelasi yang bisa disalahartikan.
- **Langkah 7: simpan hasil**
  Grafik yang bagus bisa menjadi bagian dari laporan atau notebook.

8.14 Kesalahan umum pemula

Ada beberapa jebakan yang sangat sering terjadi saat belajar visualisasi.

- **Membuat grafik sebelum tahu pertanyaan analisisnya**
- **Memilih grafik yang tidak sesuai**
  Misalnya memakai *line plot* untuk kategori acak
- **Tidak memberi label sumbu**
- **Terlalu banyak warna atau dekorasi**
- **Menganggap korelasi sebagai kausalitas**
- **Mengabaikan outlier**
- **Membaca histogram tanpa memperhatikan jumlah bins**
- **Membuat grafik cantik tetapi tidak informatif**

Jain menekankan pentingnya label, anotasi, dan tema yang jelas, tetapi juga mengingatkan bahwa visualisasi harus tetap mudah ditafsirkan.

8.15 Refleksi: visualisasi adalah alat berpikir

Bab ini seharusnya mengubah cara pandang pembaca terhadap grafik. Grafik bukan sekadar pelengkap presentasi. Grafik adalah **alat berpikir**. Dengan visualisasi, kita bisa:

- melihat pola yang tidak tampak di tabel
- mengenali anomali
- menguji dugaan awal
- membangun pertanyaan lanjutan
- menyampaikan temuan dengan lebih jelas

Outline utama buku ini sangat tepat ketika menamai bab ini **"Visualisasi Data yang Membantu Berpikir"**, bukan sekadar "membuat grafik". Itu menegaskan bahwa tujuan visualisasi bukan estetika semata, tetapi membantu proses analitis dan *storytelling* yang jujur.

Dalam konteks yang lebih luas, kemampuan visualisasi akan menjadi fondasi kuat untuk:

- EDA
- statistik praktis
- pemilihan fitur
- evaluasi model
- komunikasi hasil ke dosen, atasan, tim, atau klien

Penutup Bab

Visualisasi data adalah **keterampilan inti** bagi siapa pun yang ingin masuk ke dunia data science, AI, dan machine learning. Melalui bab ini, pembaca telah belajar membuat dan membaca **line plot**, **bar plot**, **histogram**, **boxplot**, **scatter plot**, dan **heatmap korelasi**, serta memahami kapan masing-masing grafik sebaiknya dipakai. Semua ini sangat sejalan dengan outline naskah utama dan dokumen referensi yang menempatkan visualisasi sebagai bagian penting dari EDA dan komunikasi insight.

Pesan terpenting dari bab ini sederhana: **grafik yang baik membantu kita berpikir lebih jernih, sedangkan grafik yang buruk bisa membuat kita percaya pada cerita yang salah**.

Tantangan kecil untuk pembaca: ubah program `visualisasi_terpadu.py` agar selain menampilkan grafik, program juga menyimpan setiap grafik ke folder `outputs/` dengan nama file yang rapi, misalnya `line_penjualan.png`, `bar_kategori.png`, dan `heatmap_korelasi.png`.

Bab 9. Statistik Deskriptif untuk Pemula

Bab ini adalah **titik penting sebelum pembaca masuk lebih jauh ke modeling**. Setelah belajar Python dasar, NumPy, Pandas, *data cleaning*, dan visualisasi, pembaca perlu memahami satu kemampuan inti: **meringkas data secara cerdas sebelum membuat model**. Di outline naskah utama, bab ini memang ditempatkan sebelum *machine learning* inti, dengan fokus pada **mean, median, modus, variansi, standar deviasi, persentil, kuartil, IQR, distribusi data, korelasi Pearson dan Spearman, serta pemahaman kapan statistik membantu sebelum modeling**. Struktur ini sangat tepat, karena statistik deskriptif membantu pembaca melihat **bentuk, pusat, sebaran, dan hubungan data** sebelum terlalu cepat melompat ke algoritma.

Materi dari dokumen referensi juga sangat mendukung arah ini. Buku **Statistics with Python** secara eksplisit menempatkan **mean, standard deviation, median, percentiles, IQR, Pearson correlation, dan Spearman correlation** di bab statistik deskriptif. Jain menambahkan contoh Python untuk **mean, median, mode, variance, standard deviation, serta Pearson dan Spearman correlation** dalam konteks EDA. Sementara Saltz menekankan bahwa statistik deskriptif adalah alat untuk memberi **gambaran ringkas atas dataset besar**, karena data nyata terlalu besar untuk dipahami hanya dengan melihat angka satu per satu.

Bab ini disusun agar **mudah dibaca oleh mahasiswa dan praktisi**, namun tetap profesional, *hands-on*, dan inspiratif. Tujuannya adalah membuat pembaca mampu:

- **menghitung mean, median, dan modus**
- **memahami variansi dan standar deviasi**
- **menggunakan persentil, kuartil, dan IQR**
- **membaca distribusi data**
- **menghitung dan menafsirkan korelasi Pearson dan Spearman**
- **memahami peran statistik deskriptif sebelum modeling**

Semua contoh memakai **source code Python lengkap** dan dirancang untuk berjalan di **Ubuntu 26.04**.

9.1 Mengapa statistik deskriptif penting sebelum modeling

Banyak pemula terlalu cepat ingin membuat model, padahal belum benar-benar memahami data yang dipakai. Akibatnya, model bisa terlihat berjalan, tetapi mereka tidak tahu:

- apakah datanya simetris atau sangat miring
- apakah ada nilai ekstrim
- apakah variabel target stabil atau sangat menyebar
- apakah dua fitur saling sangat terkait
- apakah hubungan antarvariabel tampak linier atau hanya berbasis peringkat

Di sinilah statistik deskriptif berperan. Referensi *descriptive analytics* menekankan bahwa statistik dasar seperti **mean, median, mode, standard deviation, dan IQR** memberi

“snapshot” tentang **tendensi pusat** dan **dispersi** data, sehingga analis bisa memahami apa yang sedang terjadi sebelum melanjutkan ke tahap yang lebih canggih.

Secara praktis, statistik deskriptif membantu menjawab pertanyaan awal berikut:

- Di mana pusat data berada?
- Seberapa lebar penyebarannya?
- Apakah data cenderung normal atau miring?
- Apakah ada *outlier*?
- Apakah dua variabel tampak berhubungan?
- Apakah data layak langsung dipakai model, atau perlu transformasi lebih dulu?

Dengan kata lain, **statistik deskriptif adalah “lampu senter” sebelum kita masuk ke ruang gelap bernama modeling**.

9.2 Dataset contoh untuk bab ini

Agar pembahasan lebih hidup, kita akan memakai dataset sederhana bertema nilai, jam belajar, dan pengeluaran belajar mahasiswa.

Simpan sebagai:

```
data/raw/statistik_mahasiswa.csv
```

Isi file:

```
nama,nilai,jam_belajar,pengeluaran_bulanan,kepuasan
Andi,78,6,450000,3
Budi,82,7,500000,4
Citra,91,9,620000,5
Dina,74,5,430000,3
Eka,88,8,580000,4
Fajar,95,10,700000,5
Gina,69,4,390000,2
Hadi,85,7,520000,4
Ika,90,9,610000,5
Joko,73,5,410000,3
Kiki,84,7,530000,4
Lina,98,11,850000,5
Mira,76,6,460000,3
Nanda,80,6,470000,4
Oki,72,5,405000,2
Prita,89,8,590000,5
Qori,77,6,455000,3
Raka,93,9,650000,5
Sinta,81,7,510000,4
Tono,65,3,350000,2
```

Kolom yang akan kita pakai:

- `nilai`
- `jam_belajar`
- `pengeluaran_bulanan`
- `kepuasan`

Dataset ini cukup baik untuk latihan statistik deskriptif karena punya data numerik, variasi nilai, dan hubungan yang cukup masuk akal untuk dianalisis.

9.3 Langkah awal: membaca dan mengenali data

Sebelum menghitung statistik apa pun, biasakan selalu:

- membaca data
- memeriksa bentuk dataset
- melihat beberapa baris awal
- memeriksa tipe data

Buat file:

```
src/statistik_awal.py
```

Isi dengan:

```
import pandas as pd

df = pd.read_csv("data/raw/statistik_mahasiswa.csv")

print("=== LIMA BARIS PERTAMA ===")
print(df.head())

print("\n=== SHAPE ===")
print(df.shape)

print("\n=== TIPE DATA ===")
print(df.dtypes)

print("\n=== RINGKASAN AWAL ===")
print(df.describe())
```

Jalankan:

```
python src/statistik_awal.py
```

Langkah sederhana ini sejalan dengan praktik EDA pada referensi, yang menyarankan pemeriksaan struktur dataset dan `describe()` sebagai langkah awal untuk memahami ukuran, tipe kolom, dan statistik ringkas.

9.4 Mean, median, modus

A. Mean

Mean atau rata-rata adalah jumlah semua nilai dibagi banyaknya observasi. Saltz menyebut mean sebagai ukuran tendensi pusat yang paling umum, yang oleh banyak orang disebut “average”.

Contoh:

```
import pandas as pd

df = pd.read_csv("data/raw/statistik_mahasiswa.csv")

mean_nilai = df["nilai"].mean()
print("Mean nilai:", mean_nilai)
```

B. Median

Median adalah nilai tengah setelah data diurutkan. Median sangat berguna ketika data memiliki *outlier* atau distribusi yang miring, karena lebih tahan terhadap nilai ekstrem dibanding mean. Saltz menekankan median sebagai ukuran pusat yang diperoleh dari posisi tengah data terurut.

```
import pandas as pd

df = pd.read_csv("data/raw/statistik_mahasiswa.csv")

median_nilai = df["nilai"].median()
print("Median nilai:", median_nilai)
```

C. Modus

Modus adalah nilai yang paling sering muncul. Jain juga memberi contoh `mode()[0]` untuk mendapatkan modus dari kolom numerik.

```
import pandas as pd

df = pd.read_csv("data/raw/statistik_mahasiswa.csv")

mode_nilai = df["nilai"].mode()[0]
print("Modus nilai:", mode_nilai)
```

D. Kapan mean, median, dan modus dipakai

Secara praktis:

- **mean** cocok untuk data yang relatif seimbang
- **median** lebih aman jika ada *outlier* atau distribusi miring
- **modus** berguna untuk melihat nilai yang paling sering muncul, terutama pada data kategori atau diskrit

E. Source code lengkap

Buat file:

```
src/mean_median_modus.py
```

Isi dengan:

```
import pandas as pd

df = pd.read_csv("data/raw/statistik_mahasiswa.csv")

for kolom in ["nilai", "jam_belajar", "pengeluaran_bulanan",
"kepuasan"]:
    print("=" * 50)
    print(f"Kolom: {kolom}")
    print("Mean   :", df[kolom].mean())
    print("Median :", df[kolom].median())
    print("Modus  :", df[kolom].mode().tolist())
```

Program ini bagus karena memperlihatkan bahwa setiap variabel bisa punya pusat data yang berbeda.

9.5 Variansi dan standar deviasi

A. Mengapa perlu ukuran sebaran

Dua dataset bisa punya mean yang sama, tetapi tingkat penyebarannya sangat berbeda. Karena itu, kita butuh ukuran **dispersi**. Jain menyebut **variance** sebagai rata-rata penyimpangan kuadrat dari mean, dan **standard deviation** sebagai akar kuadrat dari variance, sehingga nilainya kembali ke satuan asli data. Saltz juga menekankan bahwa standar deviasi lebih mudah dijelaskan ke orang awam daripada variansi, karena tidak memakai satuan kuadrat.

B. Variansi

```
import pandas as pd

df = pd.read_csv("data/raw/statistik_mahasiswa.csv")

var_nilai = df["nilai"].var()
print("Variansi nilai:", var_nilai)
```

C. Standar deviasi

```
import pandas as pd

df = pd.read_csv("data/raw/statistik_mahasiswa.csv")

std_nilai = df["nilai"].std()
print("Standar deviasi nilai:", std_nilai)
```

D. Cara membacanya

- **variansi besar** → data lebih menyebar
- **standar deviasi besar** → nilai-nilai cenderung lebih jauh dari rata-rata
- **standar deviasi kecil** → data lebih rapat di sekitar mean

E. Source code lengkap

Buat file:

```
src/variansi_std.py
```

Isi dengan:

```
import pandas as pd

df = pd.read_csv("data/raw/statistik_mahasiswa.csv")

for kolom in ["nilai", "jam_belajar", "pengeluaran_bulanan",
"kepuasan"]:
    print("=" * 50)
    print(f"Kolom: {kolom}")
    print("Minimum           :", df[kolom].min())
    print("Maksimum          :", df[kolom].max())
    print("Range             :", df[kolom].max() - df[kolom].min())
    print("Variansi          :", df[kolom].var())
    print("Standar deviasi   :", df[kolom].std())
```

Referensi *descriptive analytics* juga menekankan bahwa **range, variance, dan standard deviation** adalah ukuran penyebaran yang penting untuk memahami variasi data sebelum analisis lanjutan.

9.6 Persentil, kuartil, dan IQR

A. Persentil

Persentil menunjukkan posisi relatif suatu nilai dalam distribusi data. Misalnya persentil ke-25 berarti 25% data berada di bawah titik itu. Buku **Statistics with Python** memasukkan percentiles sebagai bagian inti dari statistik deskriptif.

Contoh:

```
import pandas as pd

df = pd.read_csv("data/raw/statistik_mahasiswa.csv")

p25 = df["nilai"].quantile(0.25)
p50 = df["nilai"].quantile(0.50)
p75 = df["nilai"].quantile(0.75)

print("Persentil 25:", p25)
print("Persentil 50:", p50)
print("Persentil 75:", p75)
```

B. Kuartil

Kuartil sebenarnya adalah persentil khusus:

- **Q1** = 25%
- **Q2** = 50% = median
- **Q3** = 75%

Kuartil membantu membagi data ke dalam empat bagian.

C. IQR

IQR (*Interquartile Range*) = Q3 - Q1. Referensi Malato menjelaskan bahwa IQR mengukur **dispersi pusat dataset dengan mengabaikan nilai ekstrem**, dan sangat berguna untuk mendeteksi *outlier* karena lebih kuat daripada standar deviasi untuk distribusi yang *skewed*.

```
import pandas as pd

df = pd.read_csv("data/raw/statistik_mahasiswa.csv")

Q1 = df["nilai"].quantile(0.25)
Q3 = df["nilai"].quantile(0.75)
IQR = Q3 - Q1

print("Q1 :", Q1)
print("Q3 :", Q3)
print("IQR:", IQR)
```

D. Menggunakan IQR untuk outlier

```
import pandas as pd

df = pd.read_csv("data/raw/statistik_mahasiswa.csv")

Q1 = df["pengeluaran_bulanan"].quantile(0.25)
Q3 = df["pengeluaran_bulanan"].quantile(0.75)
IQR = Q3 - Q1

batas_bawah = Q1 - 1.5 * IQR
batas_atas = Q3 + 1.5 * IQR

outlier = df[
    (df["pengeluaran_bulanan"] < batas_bawah) |
    (df["pengeluaran_bulanan"] > batas_atas)
]

print("Q1:", Q1)
print("Q3:", Q3)
print("IQR:", IQR)
print("Batas bawah:", batas_bawah)
print("Batas atas :", batas_atas)
print("\nData yang terdeteksi sebagai outlier:")
print(outlier)
```

Referensi Malato menuliskan aturan praktis IQR: nilai dianggap outlier jika berada di luar rentang **Q1 − 1.5 × IQR** sampai **Q3 + 1.5 × IQR**.

E. Kapan IQR berguna

IQR sangat membantu ketika:

- data tidak simetris
- ada kemungkinan nilai ekstrim
- kita butuh ukuran sebaran yang lebih tahan terhadap *outlier*

9.7 Distribusi data

A. Apa itu distribusi

Distribusi data menjelaskan **bagaimana nilai-nilai tersebar**. Saat mempelajari distribusi, kita ingin tahu:

- apakah data simetris
- apakah data miring ke kanan atau kiri
- apakah ada ekor panjang
- apakah nilai terkonsentrasi di tengah
- apakah ada beberapa puncak

Jain menyinggung *skewness* sebagai ukuran asimetri distribusi, sedangkan visualisasi seperti histogram dan boxplot sangat membantu melihat distribusi secara intuitif.

B. Histogram untuk distribusi

```
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns

df = pd.read_csv("data/raw/statistik_mahasiswa.csv")

plt.figure(figsize=(8, 5))
sns.histplot(df["nilai"], bins=8, kde=True)
plt.title("Distribusi Nilai Mahasiswa")
plt.xlabel("Nilai")
plt.ylabel("Frekuensi")
plt.tight_layout()
plt.show()
```

C. Boxplot untuk distribusi dan outlier

```
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns

df = pd.read_csv("data/raw/statistik_mahasiswa.csv")

plt.figure(figsize=(8, 5))
sns.boxplot(y=df["pengeluaran_bulanan"])
plt.title("Boxplot Pengeluaran Bulanan")
plt.ylabel("Pengeluaran Bulanan")
plt.tight_layout()
plt.show()
```

D. Skewness

Jain memberi contoh `skew()` untuk mengukur asimetri distribusi.

```
import pandas as pd

df = pd.read_csv("data/raw/statistik_mahasiswa.csv")

print("Skewness nilai             :", df["nilai"].skew())
print("Skewness pengeluaran       :",
df["pengeluaran_bulanan"].skew())
print("Skewness jam belajar       :", df["jam_belajar"].skew())
```

E. Cara membaca distribusi

Secara praktis:

- **skewness positif** → ekor lebih panjang di kanan

- **skewness negatif** → ekor lebih panjang di kiri
- **skewness mendekati nol** → distribusi lebih simetris

Distribusi membantu pembaca menentukan apakah:

- mean masih representatif
- median lebih tepat dipakai
- perlu transformasi data
- ada outlier yang perlu diperiksa

9.8 Korelasi Pearson dan Spearman

A. Mengapa korelasi penting

Korelasi membantu kita memahami **kekuatan dan arah hubungan dua variabel**. Dalam EDA, ini sangat berguna untuk melihat apakah ada pola ketergantungan yang mungkin relevan untuk analisis lebih lanjut atau modeling. Jain menyebut correlation analysis sebagai teknik kunci untuk memahami dependencies dalam data.

B. Pearson correlation

Menurut Jain, **Pearson correlation** mengukur **hubungan linier** antara dua variabel kontinu dan nilainya berada antara -1 sampai 1.

Contoh:

```
import pandas as pd

df = pd.read_csv("data/raw/statistik_mahasiswa.csv")

pearson_corr = df["jam_belajar"].corr(df["nilai"],
method="pearson")
print("Pearson correlation (jam_belajar vs nilai):",
pearson_corr)
```

C. Spearman correlation

Jain menjelaskan bahwa **Spearman correlation** mengukur **hubungan monotonik berdasarkan peringkat data**, sehingga cocok ketika hubungan tidak sepenuhnya linier atau kita lebih tertarik pada urutan naik-turun. Malato juga memberi contoh penggunaan `scipy.stats.spearmanr()` untuk menghitung koefisien Spearman.

```
import pandas as pd

df = pd.read_csv("data/raw/statistik_mahasiswa.csv")

spearman_corr = df["jam_belajar"].corr(df["nilai"],
method="spearman")
print("Spearman correlation (jam_belajar vs nilai):",
spearman_corr)
```

D. Kapan pakai Pearson, kapan Spearman

Secara praktis:

- pakai **Pearson** jika ingin melihat hubungan linier
- pakai **Spearman** jika data lebih cocok dibaca berdasarkan peringkat atau hubungan monotonik
- Spearman lebih tahan terhadap hubungan nonlinier yang tetap konsisten arahnya

E. Korelasi matriks

Kita juga bisa melihat korelasi beberapa variabel sekaligus.

```
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns

df = pd.read_csv("data/raw/statistik_mahasiswa.csv")

corr_matrix = df[["nilai", "jam_belajar", "pengeluaran_bulanan",
"kepuasan"]].corr(method="pearson")

print(corr_matrix)

plt.figure(figsize=(8, 6))
sns.heatmap(corr_matrix, annot=True, cmap="coolwarm")
plt.title("Heatmap Korelasi Pearson")
plt.tight_layout()
plt.show()
```

Jain juga mencontohkan *correlation matrix heatmap* sebagai cara berguna untuk melihat hubungan banyak variabel sekaligus.

F. Cara membaca korelasi

- mendekati **1** → hubungan positif kuat
- mendekati **-1** → hubungan negatif kuat
- mendekati **0** → hubungan linier lemah

Tetapi ingat: **korelasi bukan kausalitas**. Ini penting sekali.

9.9 Source code Python lengkap: statistik deskriptif terpadu

Berikut satu program lengkap yang menggabungkan seluruh materi bab ini.

Simpan sebagai:

```
src/statistik_deskriptif_terpadu.py
```

Isi dengan:

```
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns

# 1. Baca data
df = pd.read_csv("data/raw/statistik_mahasiswa.csv")

print("=" * 70)
print("STATISTIK DESKRIPTIF UNTUK PEMULA")
print("=" * 70)

# 2. Ringkasan awal
print("\n[1] DATA AWAL")
print(df.head())

print("\n[2] SHAPE DAN TIPE DATA")
print("Shape:", df.shape)
print(df.dtypes)

# 3. Mean, median, modus
print("\n[3] MEAN, MEDIAN, MODUS")
for kolom in ["nilai", "jam_belajar", "pengeluaran_bulanan", "kepuasan"]:
    print(f"\nKolom: {kolom}")
    print("Mean   :", df[kolom].mean())
    print("Median :", df[kolom].median())
    print("Modus  :", df[kolom].mode().tolist())

# 4. Variansi dan standar deviasi
print("\n[4] VARIANSI DAN STANDAR DEVIASI")
for kolom in ["nilai", "jam_belajar", "pengeluaran_bulanan", "kepuasan"]:
    print(f"\nKolom: {kolom}")
    print("Range            :", df[kolom].max() - df[kolom].min())
    print("Variansi         :", df[kolom].var())
    print("Standar deviasi  :", df[kolom].std())

# 5. Persentil, kuartil, IQR
print("\n[5] PERSENTIL, KUARTIL, IQR")
for kolom in ["nilai", "pengeluaran_bulanan"]:
    Q1 = df[kolom].quantile(0.25)
    Q2 = df[kolom].quantile(0.50)
    Q3 = df[kolom].quantile(0.75)
    IQR = Q3 - Q1

    print(f"\nKolom: {kolom}")
    print("Q1  :", Q1)
    print("Q2  :", Q2)
    print("Q3  :", Q3)
    print("IQR :", IQR)
```

```
# 6. Skewness distribusi
print("\n[6] DISTRIBUSI DAN SKEWNESS")
for kolom in ["nilai", "jam_belajar", "pengeluaran_bulanan"]:
    print(f"{kolom}: skewness = {df[kolom].skew()}")

# 7. Korelasi Pearson dan Spearman
print("\n[7] KORELASI")
pearson_corr = df["jam_belajar"].corr(df["nilai"], method="pearson")
spearman_corr = df["jam_belajar"].corr(df["nilai"], method="spearman")

print("Pearson  (jam_belajar vs nilai):", pearson_corr)
print("Spearman (jam_belajar vs nilai):", spearman_corr)

# 8. Korelasi matriks
corr_matrix = df[["nilai", "jam_belajar", "pengeluaran_bulanan",
"kepuasan"]].corr(method="pearson")
print("\nMatriks korelasi:")
print(corr_matrix)

# 9. Visualisasi distribusi
plt.figure(figsize=(8, 5))
sns.histplot(df["nilai"], bins=8, kde=True)
plt.title("Distribusi Nilai Mahasiswa")
plt.xlabel("Nilai")
plt.ylabel("Frekuensi")
plt.tight_layout()
plt.show()

plt.figure(figsize=(8, 5))
sns.boxplot(y=df["pengeluaran_bulanan"])
plt.title("Boxplot Pengeluaran Bulanan")
plt.ylabel("Pengeluaran Bulanan")
plt.tight_layout()
plt.show()

# 10. Heatmap korelasi
plt.figure(figsize=(8, 6))
sns.heatmap(corr_matrix, annot=True, cmap="coolwarm")
plt.title("Heatmap Korelasi Pearson")
plt.tight_layout()
plt.show()

print("\nSELESAI. Anda sudah menjalankan statistik deskriptif inti.")
print("=" * 70)
```

Jalankan dengan:

```
python src/statistik_deskriptif_terpadu.py
```

Program ini sangat baik untuk menggabungkan semua konsep inti dalam satu latihan praktik.

9.10 Kapan statistik membantu sebelum modeling

Ini bagian yang paling penting dari bab ini.

Sebelum modeling, statistik deskriptif membantu kita memutuskan:

- apakah data relatif bersih dan masuk akal
- apakah ada *outlier* yang perlu ditangani
- apakah distribusi target sangat miring
- apakah skala fitur sangat berbeda
- apakah ada fitur yang sangat berkorelasi
- apakah mean masih representatif, atau median lebih aman
- apakah hubungan antar variabel tampak linier atau lebih cocok dibaca dengan peringkat

Referensi *descriptive analytics* menegaskan bahwa statistik dasar membantu menjawab pertanyaan: **apa yang terjadi, seberapa sering, dan seberapa besar dampaknya**, sehingga memberi fondasi untuk langkah analitik berikutnya.

Contoh konkret

Misalnya sebelum membuat model prediksi nilai:

- kalau `pengeluaran_bulanan` sangat *skewed*, mungkin perlu transformasi
- kalau `jam_belajar` dan `nilai` berkorelasi kuat, itu sinyal fitur yang masuk akal
- kalau ada *outlier* ekstrem, model bisa tertarik ke arah yang salah
- kalau `kepuasan` hanya skala ordinal 1–5, mungkin Spearman lebih relevan daripada Pearson

Singkatnya

> **Statistik deskriptif membantu kita memahami "bahan baku" sebelum memasukkan data ke mesin model.**

Tanpa langkah ini, modeling sering menjadi sekadar "tekan tombol", bukan analisis yang sadar.

9.11 Urutan kerja yang disarankan

Agar pembaca membangun kebiasaan yang benar, berikut urutan kerja yang saya sarankan setiap kali menerima dataset baru:

- **Langkah 1:** baca data dan cek struktur
- **Langkah 2:** hitung mean, median, modus
- **Langkah 3:** hitung range, variansi, dan standar deviasi
- **Langkah 4:** cek persentil, kuartil, dan IQR
- **Langkah 5:** lihat distribusi dengan histogram dan boxplot
- **Langkah 6:** hitung korelasi Pearson dan Spearman bila perlu
- **Langkah 7:** simpulkan apakah data siap masuk ke tahap modeling atau masih perlu pembersihan/transformasi

Urutan ini sejalan dengan materi lampiran yang menempatkan statistik deskriptif dan analisis korelasi sebagai inti EDA sebelum model dibangun.

9.12 Kesalahan umum pemula

Ada beberapa jebakan yang sangat sering terjadi:

- **mengandalkan mean saja**, padahal data mungkin punya *outlier*
- **melihat korelasi tinggi lalu menganggap ada sebab-akibat**
- **tidak memeriksa distribusi**, padahal bentuk data sangat memengaruhi interpretasi
- **memakai Pearson di semua kasus**, padahal kadang Spearman lebih masuk akal
- **tidak mengecek IQR atau boxplot**, sehingga outlier luput
- **langsung modeling tanpa statistik awal**

Kebiasaan yang lebih baik adalah selalu bertanya:

> **apa yang sebenarnya sedang diceritakan data ini, sebelum saya membuat model?**

9.13 Refleksi: statistik deskriptif adalah bahasa pertama data

Bab ini mengajarkan sesuatu yang sangat mendasar: sebelum data “diprediksi”, data harus lebih dulu **dipahami**. Statistik deskriptif adalah bahasa pertama untuk memahami data. Dengan mean, median, modus, variansi, standar deviasi, persentil, IQR, distribusi, dan korelasi, pembaca mulai bisa membaca karakter data seperti seorang analis, bukan hanya sebagai pengguna kode.

Itulah sebabnya statistik deskriptif sangat penting sebelum modeling. Ia membuat pembaca lebih hati-hati, lebih sadar konteks, dan lebih paham kualitas data. Seperti ditekankan dalam referensi, deskriptif statistik membantu merangkum data besar menjadi gambaran yang lebih bisa dipahami, dan itulah fondasi untuk keputusan berbasis data.

Penutup Bab

Statistik deskriptif adalah **alat inti** bagi setiap pemula yang ingin serius masuk ke dunia data science, AI, dan machine learning. Melalui bab ini, pembaca telah belajar tentang **mean, median, modus, variansi, standar deviasi, persentil, kuartil, IQR, distribusi data, serta korelasi Pearson dan Spearman**, dan yang paling penting, memahami **kapan statistik membantu sebelum modeling**. Semua ini sangat selaras dengan outline naskah utama dan dokumen referensi yang menempatkan statistik deskriptif sebagai fondasi EDA dan analisis awal.

Pesan terpenting dari bab ini sederhana: **sebelum membuat model yang canggih, pahami dulu cerita dasar yang ada di data**.

Tantangan kecil untuk pembaca: modifikasi program `statistik_deskriptif_terpadu.py` agar selain menghitung statistik untuk `nilai`, program juga membandingkan statistik `nilai` antara dua kelompok: mahasiswa dengan `jam_belajar >= 7` dan mahasiswa dengan `jam_belajar < 7`.

Bab 10. Regresi Linear sebagai Jembatan ke Machine Learning

Bab ini adalah **jembatan yang sangat penting** antara statistik deskriptif dan *machine learning* yang sesungguhnya. Di titik ini, pembaca tidak lagi hanya merangkum data, tetapi mulai belajar **membangun model yang mencoba memprediksi nilai numerik**. Materi ini memang sangat kuat didukung dokumen lampiran, terutama yang membahas **linear regression**, penggunaan **NumPy, Pandas, Matplotlib**, dan praktik evaluasi model dengan **MSE** serta **R^2**. Beberapa referensi juga menekankan bahwa regresi linear adalah model yang sangat baik untuk pemula karena **sederhana, interpretatif, dan menjadi pintu masuk alami ke predictive modeling**.

Secara konseptual, regresi linear membantu pembaca memahami hal-hal yang nantinya akan terus muncul di *machine learning*: **fitur**, **target**, **koefisien**, **intercept**, **prediksi**, **error**, **train-test split**, dan **evaluasi model**. Dalam referensi pendukung, regresi linear dijelaskan sebagai model yang mencoba mencari garis lurus terbaik yang menggambarkan hubungan antara variabel input dan output, baik pada kasus satu fitur maupun banyak fitur. Pendekatan ini juga menekankan bahwa koefisien memberi interpretasi langsung terhadap pengaruh fitur terhadap target.

Bab ini disusun agar **mudah dibaca oleh mahasiswa dan praktisi**, tetapi tetap profesional, aplikatif, dan menantang. Targetnya adalah membuat pembaca mampu:

- **memahami intuisi hubungan antar variabel**
- **membangun regresi linear sederhana**
- **membangun regresi linear berganda**
- **membaca koefisien, intercept, dan residual**
- **memvisualisasikan garis regresi**
- **mengevaluasi model dengan MAE, MSE, RMSE, dan R^2**
- **menggunakan scikit-learn untuk alur kerja regresi linear yang rapi**

10.1 Mengapa regresi linear penting

Regresi linear penting bukan hanya karena ia model klasik, tetapi karena ia mengajarkan **cara berpikir prediktif yang paling dasar dan paling jernih**. Dalam banyak kasus nyata, kita ingin memprediksi nilai numerik seperti:

- harga rumah
- omzet penjualan
- konsumsi energi
- gaji
- nilai mahasiswa
- biaya operasional

Untuk kasus seperti itu, regresi linear menjadi salah satu model pertama yang paling masuk akal dicoba. Referensi pendukung menekankan bahwa regresi linear adalah salah satu algoritma paling sederhana dan paling banyak dipakai, karena ia memodelkan hubungan

antara target dan satu atau lebih fitur melalui persamaan linear yang mudah diinterpretasikan.

Dari sisi pembelajaran, regresi linear sangat berharga karena:

- **mudah dipahami**
- **mudah divisualisasikan**
- **hasilnya bisa dijelaskan**
- **cocok untuk mengajarkan evaluasi error**
- **menjadi fondasi menuju model yang lebih kompleks**

Dengan kata lain, regresi linear bukan hanya algoritma, tetapi **alat latihan berpikir model**.

10.2 Intuisi hubungan antar variabel

Sebelum menulis kode, pembaca perlu memahami **intuisi hubungan antar variabel**. Dalam regresi linear, kita bertanya:

> **apakah perubahan pada variabel X berhubungan dengan perubahan pada variabel Y?**

Misalnya:

- semakin besar **jam belajar**, apakah **nilai** cenderung naik?
- semakin besar **biaya iklan**, apakah **penjualan** cenderung naik?
- semakin luas **rumah**, apakah **harga** cenderung naik?

Referensi menjelaskan bahwa regresi linear pada dasarnya mencoba menggambar **garis lurus terbaik** yang mewakili hubungan antara variabel input dan output. Garis ini disebut **regression line**, dan bentuk sederhananya adalah:

```
y = mx + c
```

di mana:

- **y** = target yang diprediksi
- **x** = fitur
- **m** = slope atau koefisien
- **c** = intercept

Intuisi sederhana

Kalau kita punya data:

Jam Belajar	Nilai
2	60
4	68
6	76
8	85
10	93

Mata kita langsung melihat pola: **semakin tinggi jam belajar, semakin tinggi nilai**. Regresi linear mencoba menangkap pola itu dengan satu garis lurus.

Hubungan tidak selalu sempurna

Dalam data nyata, titik-titik tidak selalu berada tepat di satu garis. Akan selalu ada selisih antara nilai sebenarnya dan nilai prediksi. Selisih inilah yang nanti disebut **residual** atau error. Referensi pendukung juga menekankan bahwa regresi linear bekerja dengan meminimalkan **sum of squared residuals**, yaitu total kuadrat selisih antara nilai aktual dan nilai prediksi.

10.3 Dataset contoh untuk bab ini

Agar pembahasan lebih hidup, kita akan memakai dua dataset sederhana:

Dataset 1 — Regresi linear sederhana

Simpan sebagai:

```
data/raw/regresi_sederhana.csv
```

Isi file:

```
jam_belajar,nilai
2,60
3,64
4,68
5,72
6,76
7,80
8,84
9,88
10,92
11,95
12,98
```

Dataset 2 — Regresi linear berganda

Simpan sebagai:

```
data/raw/regresi_berganda.csv
```

Isi file:

```
jam_belajar,kehadiran,tugas,nilai
2,70,60,58
3,75,65,63
4,78,68,67
5,80,70,71
6,82,75,75
7,85,78,79
8,88,82,84
9,90,85,88
10,92,88,91
11,95,90,94
12,96,94,97
```

Dataset ini sengaja dibuat sederhana supaya pembaca bisa fokus pada konsep dan alur kerja model.

10.4 Regresi linear sederhana

Konsep

Regresi linear sederhana berarti kita hanya memakai **satu fitur** untuk memprediksi satu target.

Contoh:

- fitur: `jam_belajar`
- target: `nilai`

Persamaannya:

```
nilai = intercept + koefisien × jam_belajar
```

Langkah kerja yang benar

Untuk pemula, alur yang sehat adalah:

- baca data
- pisahkan fitur `X` dan target `y`
- buat model
- latih model dengan `fit()`
- buat prediksi
- lihat koefisien dan intercept
- evaluasi hasil
- visualisasikan garis regresi

Source code lengkap

Simpan sebagai:

```
src/regresi_linear_sederhana.py
```

Isi file:

```
import pandas as pd
import matplotlib.pyplot as plt
from sklearn.linear_model import LinearRegression
from sklearn.metrics import mean_absolute_error, mean_squared_error,
r2_score
import math

# 1. Baca data
df = pd.read_csv("data/raw/regresi_sederhana.csv")

print("=== DATA AWAL ===")
print(df)

# 2. Pisahkan fitur dan target
X = df[["jam_belajar"]]
y = df["nilai"]

# 3. Buat model
model = LinearRegression()

# 4. Latih model
model.fit(X, y)

# 5. Prediksi
y_pred = model.predict(X)

# 6. Ambil parameter model
koefisien = model.coef_[0]
intercept = model.intercept_

print("\n=== PARAMETER MODEL ===")
print("Koefisien :", koefisien)
print("Intercept :", intercept)
```

```
# 7. Hitung residual
df["prediksi"] = y_pred
df["residual"] = df["nilai"] - df["prediksi"]

print("\n=== DATA DENGAN PREDIKSI DAN RESIDUAL ===")
print(df)

# 8. Evaluasi model
mae = mean_absolute_error(y, y_pred)
mse = mean_squared_error(y, y_pred)
rmse = math.sqrt(mse)
r2 = r2_score(y, y_pred)

print("\n=== EVALUASI MODEL ===")
print("MAE  :", mae)
print("MSE  :", mse)
print("RMSE :", rmse)
print("R²   :", r2)

# 9. Visualisasi garis regresi
plt.figure(figsize=(8, 5))
plt.scatter(X, y, label="Data aktual")
plt.plot(X, y_pred, label="Garis regresi")
plt.title("Regresi Linear Sederhana: Jam Belajar vs Nilai")
plt.xlabel("Jam Belajar")
plt.ylabel("Nilai")
plt.legend()
plt.grid(True)
plt.tight_layout()
plt.show()
```

Referensi pendukung memberi pola yang sangat serupa: membaca data ke DataFrame, menentukan `X` dan `y`, memakai `LinearRegression()`, melakukan `fit()`, membuat `predict()`, lalu menghitung **MSE** dan menampilkan **koefisien** serta **intercept**. Visualisasi garis regresi di atas titik data juga ditekankan sebagai cara yang sangat baik untuk membantu pemahaman.

10.5 Koefisien, intercept, dan residual

Ini adalah inti interpretasi model.

A. Koefisien

Koefisien menunjukkan **seberapa besar perubahan target untuk setiap kenaikan satu unit fitur**, dengan asumsi hal lain tetap.

Kalau hasil model memberi:

- koefisien = 3.4

maka artinya:

> **setiap tambahan 1 jam belajar diperkirakan menaikkan nilai sekitar 3.4 poin**

Referensi pendukung menjelaskan bahwa slope atau koefisien menunjukkan perubahan nilai prediksi ketika fitur naik satu satuan.

B. Intercept

Intercept adalah nilai prediksi target ketika semua fitur bernilai nol.

Kalau:

- intercept = 52

maka artinya:

jika jam_belajar = 0, model memprediksi nilai sekitar 52

Tentu secara konteks, intercept tidak selalu harus ditafsirkan secara literal. Kadang ia hanya parameter matematis untuk menempatkan garis pada posisi terbaik.

C. Residual

Residual adalah:

nilai aktual - nilai prediksi

Residual menunjukkan seberapa jauh model meleset pada setiap observasi. Referensi juga menyebut residual sebagai komponen error yang menangkap perbedaan antara nilai aktual dan nilai model.

Kalau residual:

- positif → model memprediksi terlalu rendah
- negatif → model memprediksi terlalu tinggi
- mendekati nol → model cukup akurat pada titik itu

Visualisasi residual sederhana

Simpan sebagai:

`src/plot_residual_sederhana.py`

Isi file:

```
import pandas as pd
import matplotlib.pyplot as plt
from sklearn.linear_model import LinearRegression

df = pd.read_csv("data/raw/regresi_sederhana.csv")

X = df[["jam_belajar"]]
y = df["nilai"]

model = LinearRegression()
model.fit(X, y)

df["prediksi"] = model.predict(X)
df["residual"] = df["nilai"] - df["prediksi"]

plt.figure(figsize=(8, 5))
plt.axhline(y=0, linestyle="--")
plt.scatter(df["prediksi"], df["residual"])
plt.title("Plot Residual")
plt.xlabel("Nilai Prediksi")
plt.ylabel("Residual")
plt.grid(True)
plt.tight_layout()
plt.show()
```

Plot residual membantu melihat apakah error tersebar cukup acak atau justru punya pola yang menunjukkan model linear kurang cocok.

10.6 Visualisasi garis regresi

Visualisasi adalah bagian yang sangat penting dalam regresi linear sederhana. Referensi pendukung menekankan bahwa menampilkan **scatter plot data aktual** dan **garis regresi** sangat membantu untuk menjelaskan hubungan linear serta efektivitas model.

Mengapa visualisasi ini penting

Dengan visualisasi, pembaca bisa langsung melihat:

- apakah titik-titik mendekati garis
- apakah hubungan tampak linear
- apakah ada titik yang terlalu jauh
- apakah model cukup masuk akal

Versi visual yang lebih rapi

Simpan sebagai:

```
src/visual_regresi_sederhana.py
```

Isi file:

```
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.linear_model import LinearRegression

df = pd.read_csv("data/raw/regresi_sederhana.csv")

X = df[["jam_belajar"]]
y = df["nilai"]

model = LinearRegression()
model.fit(X, y)
y_pred = model.predict(X)

plt.figure(figsize=(8, 5))
sns.scatterplot(data=df, x="jam_belajar", y="nilai", s=80)
plt.plot(df["jam_belajar"], y_pred)
plt.title("Visualisasi Regresi Linear Sederhana")
plt.xlabel("Jam Belajar")
plt.ylabel("Nilai")
plt.grid(True)
plt.tight_layout()
plt.show()
```

10.7 Regresi linear berganda

Konsep

Regresi linear berganda berarti kita memakai **lebih dari satu fitur** untuk memprediksi target.

Contoh:

- `jam_belajar`
- `kehadiran`
- `tugas`

untuk memprediksi:

- `nilai`

Persamaannya menjadi:

`y = b0 + b1x1 + b2x2 + b3x3 + ... + bp xp`

Dokumen referensi menjelaskan bentuk umum regresi linear berganda seperti ini, dengan `b0` sebagai intercept, `b1...bp` sebagai koefisien, dan residual/error sebagai selisih antara nilai aktual dan prediksi.

Intuisi

Kalau regresi sederhana hanya bertanya:

apakah X memengaruhi Y?

maka regresi berganda bertanya:

bagaimana beberapa faktor bersama-sama berhubungan dengan Y?

Ini lebih realistis, karena dalam dunia nyata target hampir selalu dipengaruhi lebih dari satu faktor.

Source code lengkap

Simpan sebagai:

`src/regresi_linear_berganda.py`

Isi file:

```
import pandas as pd
from sklearn.linear_model import LinearRegression
from sklearn.metrics import mean_absolute_error, mean_squared_error,
r2_score
import math

# 1. Baca data
df = pd.read_csv("data/raw/regresi_berganda.csv")

print("=== DATA AWAL ===")
print(df)

# 2. Pisahkan fitur dan target
X = df[["jam_belajar", "kehadiran", "tugas"]]
y = df["nilai"]

# 3. Buat model
model = LinearRegression()

# 4. Latih model
model.fit(X, y)

# 5. Prediksi
y_pred = model.predict(X)

# 6. Parameter
print("\n=== PARAMETER MODEL ===")
print("Intercept:", model.intercept_)
print("Koefisien:")
for nama_fitur, coef in zip(X.columns, model.coef_):
    print(f"{nama_fitur}: {coef}")

# 7. Residual
df["prediksi"] = y_pred
df["residual"] = df["nilai"] - df["prediksi"]
```

```
print("\n=== DATA DENGAN PREDIKSI DAN RESIDUAL ===")
print(df)

# 8. Evaluasi
mae = mean_absolute_error(y, y_pred)
mse = mean_squared_error(y, y_pred)
rmse = math.sqrt(mse)
r2 = r2_score(y, y_pred)

print("\n=== EVALUASI MODEL ===")
print("MAE  :", mae)
print("MSE  :", mse)
print("RMSE :", rmse)
print("R²   :", r2
```

Referensi pendukung juga mencontohkan regresi berganda dengan fitur seperti `Square_Feet` dan `Bedrooms` untuk memprediksi `Price`, menggunakan `LinearRegression`, `train_test_split`, dan `mean_squared_error`, lalu menampilkan koefisien dan intercept.

10.8 Evaluasi dasar: MAE, MSE, RMSE, dan R²

Setelah model dibuat, pekerjaan belum selesai. Model harus **dievaluasi**. Ini sangat ditekankan dalam referensi: model tidak cukup hanya dijalankan, tetapi perlu diukur error dan daya jelaskannya.

A. MAE — Mean Absolute Error

MAE adalah rata-rata **nilai absolut error**.

Maknanya:

- semakin kecil, semakin baik
- mudah dipahami karena tetap dalam satuan asli target

```
from sklearn.metrics import mean_absolute_error
mae = mean_absolute_error(y, y_pred)
```

B. MSE — Mean Squared Error

MSE adalah rata-rata **error kuadrat**. Referensi menjelaskan bahwa MSE memberi penalti lebih besar untuk error besar, sehingga berguna ketika kesalahan besar dianggap lebih serius.

```
from sklearn.metrics import mean_squared_error
mse = mean_squared_error(y, y_pred)
```

C. RMSE — Root Mean Squared Error

RMSE adalah akar dari MSE. Referensi menekankan bahwa RMSE kembali ke satuan asli target, sehingga lebih mudah diinterpretasikan daripada MSE.

```
import math
rmse = math.sqrt(mse)
```

D. R^2 — Coefficient of Determination

R^2 mengukur seberapa besar variasi target yang bisa dijelaskan model. Referensi menjelaskan bahwa R^2 berada antara 0 dan 1 dalam konteks umum, dan semakin tinggi berarti model menjelaskan variasi target dengan lebih baik.

```
from sklearn.metrics import r2_score
r2 = r2_score(y, y_pred)
```

Cara membaca metrik

Secara praktis:

- **MAE kecil** → rata-rata kesalahan kecil
- **MSE kecil** → error besar jarang terjadi
- **RMSE kecil** → prediksi dekat dengan data aktual
- **R^2 tinggi** → model menjelaskan variasi target dengan cukup baik

Catatan penting

Tidak ada angka “baik” yang universal. Interpretasi tetap harus melihat konteks data dan satuan target.

10.9 Train-test split agar lebih realistis

Pada contoh awal, kita melatih dan menguji model pada data yang sama. Untuk belajar konsep dasar itu boleh, tetapi untuk pendekatan *machine learning* yang lebih sehat, kita perlu memisahkan data latih dan data uji.

Dokumen lampiran berkali-kali menekankan penggunaan `train_test_split` sebagai bagian penting dari alur kerja model.

Source code lengkap

Simpan sebagai:

```
src/regresi_dengan_train_test_split.py
```

Isi file:

```
import pandas as pd
import math
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LinearRegression
from sklearn.metrics import mean_absolute_error, mean_squared_error,
r2_score
```

```
df = pd.read_csv("data/raw/regresi_berganda.csv")

X = df[["jam_belajar", "kehadiran", "tugas"]]
y = df["nilai"]

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

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

# Prediksi pada data uji
y_pred = model.predict(X_test)

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

print("=== HASIL TRAIN-TEST SPLIT ===")
print("MAE  :", mae)
print("MSE  :", mse)
print("RMSE :", rmse)
print("R²   :", r2)

print("\nIntercept:", model.intercept_)
print("Koefisien:")
for fitur, coef in zip(X.columns, model.coef_):
    print(f"{fitur}: {coef}")

hasil = pd.DataFrame({
    "aktual": y_test.values,
    "prediksi": y_pred
})

print("\n=== PERBANDINGAN AKTUAL VS PREDIKSI ===")
print(hasil)
```

10.10 Contoh interpretasi hasil model

Ini bagian yang sering diabaikan pemula. Padahal model tidak berhenti pada angka metrik. Model perlu **dibaca**.

Misalnya model berganda memberi hasil:

- intercept = 10
- koefisien jam_belajar = 2.5
- koefisien kehadiran = 0.3
- koefisien tugas = 0.4

Maka interpretasi sederhananya:

- setiap tambahan 1 jam belajar, nilai diperkirakan naik 2.5 poin, dengan asumsi fitur lain tetap
- setiap tambahan 1 poin kehadiran, nilai naik sekitar 0.3
- setiap tambahan 1 poin tugas, nilai naik sekitar 0.4

Dari sini pembaca mulai melihat bahwa model linear tidak hanya memprediksi, tetapi juga **membantu menjelaskan hubungan antar variabel**. Inilah salah satu kekuatan utama regresi linear dibanding beberapa model yang lebih kompleks: ia lebih mudah diinterpretasikan. Referensi lampiran juga secara eksplisit menyebut bahwa linear regression memiliki keunggulan interpretabilitas yang penting untuk menjelaskan hasil ke pemangku kepentingan.

10.11 Asumsi praktis dan keterbatasan regresi linear

Meskipun sangat berguna, regresi linear tidak selalu cocok. Referensi pendukung menyinggung bahwa linear regression mengasumsikan hubungan linear, dan bisa gagal jika pola data sebenarnya kompleks atau nonlinier.

Regresi linear cocok jika:

- hubungan fitur-target cukup linear
- data tidak terlalu liar
- interpretasi model penting
- baseline model diperlukan

Regresi linear kurang cocok jika:

- hubungan sangat nonlinier
- ada pola interaksi kompleks
- banyak outlier berat
- fitur saling sangat berkorelasi tinggi
- error residual menunjukkan pola kuat

Tanda model linear mungkin kurang cocok

- scatter plot tidak tampak linear
- residual punya pola tertentu
- error tinggi
- R^2 rendah
- model lain jauh lebih baik

Tetapi untuk pembelajaran dan baseline, regresi linear tetap sangat penting.

10.12 Source code Python lengkap: latihan terpadu Bab 10

Berikut satu program lengkap yang merangkum inti bab ini: regresi sederhana, regresi berganda, evaluasi, koefisien, intercept, residual, dan visualisasi.

Simpan sebagai:

```
src/bab10_regresi_linear_terpadu.py
```

Isi file:

```
import pandas as pd
import matplotlib.pyplot as plt
import math
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LinearRegression
from sklearn.metrics import mean_absolute_error, mean_squared_error,
r2_score

print("=" * 70)
print("BAB 10 - REGRESI LINEAR SEBAGAI JEMBATAN KE MACHINE LEARNING")
print("=" * 70)

# ============================================================
# BAGIAN A - REGRESI LINEAR SEDERHANA
# ============================================================
print("\n[ A ] REGRESI LINEAR SEDERHANA")

df1 = pd.read_csv("data/raw/regresi_sederhana.csv")
X1 = df1[["jam_belajar"]]
y1 = df1["nilai"]

model1 = LinearRegression()
model1.fit(X1, y1)
y1_pred = model1.predict(X1)

print("Intercept :", model1.intercept_)
print("Koefisien :", model1.coef_[0])

mae1 = mean_absolute_error(y1, y1_pred)
mse1 = mean_squared_error(y1, y1_pred)
rmse1 = math.sqrt(mse1)
```

```
r21 = r2_score(y1, y1_pred)

print("MAE  :", mae1)
print("MSE  :", mse1)
print("RMSE :", rmse1)
print("R²   :", r21)

df1["prediksi"] = y1_pred
df1["residual"] = df1["nilai"] - df1["prediksi"]

print("\nData dengan residual:")
print(df1)

plt.figure(figsize=(8, 5))
plt.scatter(X1, y1, label="Data aktual")
plt.plot(X1, y1_pred, label="Garis regresi")
plt.title("Regresi Linear Sederhana")
plt.xlabel("Jam Belajar")
plt.ylabel("Nilai")
plt.legend()
plt.grid(True)
plt.tight_layout()
plt.show()

# ============================================================
# BAGIAN B - REGRESI LINEAR BERGANDA
# ============================================================
print("\n[ B ] REGRESI LINEAR BERGANDA")

df2 = pd.read_csv("data/raw/regresi_berganda.csv")
X2 = df2[["jam_belajar", "kehadiran", "tugas"]]
y2 = df2["nilai"]

X_train, X_test, y_train, y_test = train_test_split(
    X2, y2, test_size=0.2, random_state=42
)

model2 = LinearRegression()
model2.fit(X_train, y_train)
y2_pred = model2.predict(X_test)

print("Intercept :", model2.intercept_)
print("Koefisien :")
for fitur, coef in zip(X2.columns, model2.coef_):
    print(f"  {fitur}: {coef}")

mae2 = mean_absolute_error(y_test, y2_pred)
mse2 = mean_squared_error(y_test, y2_pred)
rmse2 = math.sqrt(mse2)
r22 = r2_score(y_test, y2_pred)

print("MAE  :", mae2)
print("MSE  :", mse2)
```

```
print("RMSE :", rmse2)
print("R²   :", r22)

hasil_test = pd.DataFrame({
    "aktual": y_test.values,
    "prediksi": y2_pred,
    "residual": y_test.values - y2_pred
})

print("\nHasil data uji:")
print(hasil_test)

print("\nSELESAI. Anda sudah membangun model regresi linear sederhana
dan berganda.")
print("=" * 70)
```

10.13 Langkah kerja yang disarankan untuk pembaca

Agar bab ini benar-benar membentuk keterampilan, berikut urutan kerja yang saya sarankan.

- **Langkah 1: pahami dulu pertanyaan prediksi**
  Apakah target berupa angka?
- **Langkah 2: pilih fitur dan target**
  Pisahkan `X` dan `y`.
- **Langkah 3: mulai dari model sederhana**
  Pakai satu fitur dulu agar hubungan mudah divisualisasikan.
- **Langkah 4: latih model**
  Gunakan `LinearRegression().fit()`.
- **Langkah 5: baca parameter model**
  Lihat `coef_` dan `intercept_`.
- **Langkah 6: buat prediksi**
  Gunakan `predict()`.
- **Langkah 7: evaluasi**
  Hitung MAE, MSE, RMSE, dan R^2.
- **Langkah 8: cek residual**
  Lihat apakah error tampak wajar.
- **Langkah 9: visualisasikan**
  Scatter plot + garis regresi untuk model sederhana.
- **Langkah 10: baru coba model berganda**
  Tambahkan lebih banyak fitur bila masuk akal.

10.14 Kesalahan umum pemula

Ada beberapa jebakan yang sangat sering terjadi.

- **langsung memakai banyak fitur tanpa paham hubungan dasarnya**
- **tidak memisahkan fitur dan target dengan benar**
- **mengira R^2 tinggi selalu berarti model bagus**
- **mengabaikan residual**
- **menganggap korelasi atau regresi sebagai bukti kausalitas**
- **menginterpretasikan intercept secara berlebihan**
- **menguji model pada data yang sama tanpa sadar bahwa itu terlalu optimistis**
- **tidak melihat konteks satuan error**

Kebiasaan yang baik adalah selalu bertanya:

> **apakah model ini masuk akal secara data, secara matematika, dan secara konteks?**

10.15 Refleksi: mengapa regresi linear adalah jembatan yang sangat baik

Bab ini penting karena ia menghubungkan dua dunia:

- dunia **statistik**, yang menekankan hubungan dan interpretasi
- dunia **machine learning**, yang menekankan prediksi dan evaluasi

Regresi linear berada tepat di tengahnya. Ia cukup matematis untuk mengajarkan konsep seperti koefisien, intercept, dan residual, tetapi juga cukup praktis untuk mengajarkan alur *machine learning* seperti `train_test_split`, `fit`, `predict`, dan metrik evaluasi. Itulah mengapa regresi linear sangat layak disebut **jembatan ke machine learning**.

Melalui bab ini, pembaca tidak hanya belajar “cara memakai scikit-learn”, tetapi juga belajar membaca hubungan antar variabel, memvisualisasikan model, dan menilai apakah prediksi yang dibuat cukup baik. Referensi lampiran juga menegaskan peran regresi linear sebagai fondasi yang kuat, interpretatif, dan sangat berguna sebelum pembaca bergerak ke model lain yang lebih kompleks.

Penutup Bab

Regresi linear adalah **pintu masuk terbaik** untuk memahami predictive modeling secara bertahap. Melalui bab ini, pembaca telah belajar **intuisi hubungan antar variabel**, **regresi linear sederhana**, **regresi linear berganda**, **koefisien, intercept, residual**, **visualisasi garis regresi**, serta **evaluasi dasar dengan MAE, MSE, RMSE, dan R^2**, semuanya dengan **contoh source code Python menggunakan scikit-learn**. Materi ini memang sangat kuat ditopang oleh dokumen lampiran yang menekankan hubungan antara linear regression, NumPy, Pandas, Matplotlib, dan evaluasi model.

Pesan terpenting dari bab ini sederhana: **sebelum masuk ke model yang lebih rumit, kuasai dulu model yang bisa Anda pahami, jelaskan, dan evaluasi dengan jernih**.

Tantangan kecil untuk pembaca: modifikasi program `bab10_regresi_linear_terpadu.py` agar model tidak hanya memakai `jam_belajar`, `kehadiran`, dan `tugas`, tetapi juga menambahkan satu fitur baru buatan Anda sendiri, misalnya `motivasi`, lalu bandingkan perubahan nilai R^2 dan RMSE.

Bab 11. Statistik Inferensial Ringan untuk Data Science

Bab ini adalah langkah penting setelah statistik deskriptif. Jika statistik deskriptif membantu kita **meringkas apa yang terlihat pada data**, maka statistik inferensial membantu kita **membuat penalaran dari sampel menuju populasi**, dengan cara yang lebih hati-hati, terukur, dan dapat dipertanggungjawabkan. Di outline naskah utama, bab ini memang ditempatkan sebagai pengantar ringan sebelum pembaca masuk lebih jauh ke *machine learning*, agar pembaca tidak menganggap analisis data hanya sekadar menghitung rata-rata atau membuat model, tetapi juga memahami **ketidakpastian, asumsi, dan makna hasil**. Arah ini sangat selaras dengan materi lampiran yang membahas **probability**, **sampling**, **central limit theorem**, **hypothesis testing**, dan **confidence-oriented interpretation**.

Materi referensi juga menunjukkan pola yang sangat konsisten. Buku **Statistics with Python** secara eksplisit bergerak dari **probability** dan **descriptive statistics** menuju **hypothesis testing** dan **confidence intervals**, lalu menekankan bahwa tujuan latihan statistik adalah menerjemahkan masalah bisnis menjadi pertanyaan statistik dan mengkomunikasikan hasil dengan benar. Sementara **An Introduction to Data Science With Python** menempatkan topik **sampling**, **law of large numbers**, **central limit theorem**, dan **statistical inference** sebagai jembatan penting dari deskripsi ke pengambilan keputusan. Ini sangat cocok untuk bab yang ingin dibuat **ringan, intuitif, dan aplikatif**.

Bab ini dirancang untuk pembaca pemula, mahasiswa, dan praktisi yang ingin memahami pertanyaan seperti:

- **apakah hasil sampel bisa mewakili populasi**
- **bagaimana probabilitas membantu membaca ketidakpastian**
- **apa arti uji statistik sederhana**
- **mengapa statistik penting untuk interpretasi, bukan sekadar hitung**

11.1 Mengapa statistik inferensial penting untuk *data science*

Dalam proyek *data science*, kita hampir selalu bekerja dengan **data yang terbatas**. Jarang sekali kita memiliki seluruh populasi. Misalnya:

- kita hanya punya **100 responden survei**, bukan seluruh pelanggan
- kita hanya punya **1.000 transaksi sampel**, bukan semua transaksi nasional
- kita hanya mengamati **beberapa minggu penjualan**, bukan seluruh masa depan
- kita hanya menguji model pada **satu subset data**, bukan semua kemungkinan data

Karena itu, kita perlu satu cara berpikir yang menjawab pertanyaan:

> **apa yang bisa kita simpulkan dari sebagian data terhadap keseluruhan keadaan?**

Itulah inti statistik inferensial. Referensi Saltz menekankan bahwa inferensi statistik muncul ketika kita bergerak dari deskripsi populasi yang terlihat menuju keputusan berdasarkan **sampling distribution** dan **thresholds**. Sementara Malato menekankan bahwa statistik

harus membantu analis menerjemahkan masalah nyata menjadi pertanyaan statistik yang relevan dan mengkomunikasikan hasil dengan benar.

Secara praktis, statistik inferensial membantu kita untuk:

- menilai apakah perbedaan yang terlihat **mungkin hanya kebetulan**
- memahami bahwa hasil data selalu punya **unsur ketidakpastian**
- membedakan antara **pola yang kuat** dan **fluktuasi acak**
- membaca hasil analisis dengan lebih dewasa

Inilah alasan mengapa bab ini penting. Dalam *data science*, statistik tidak boleh berhenti di "berapa rata-ratanya?", tetapi harus lanjut ke "seberapa yakin kita terhadap hasil ini?"

11.2 Sampel vs populasi

A. Apa itu populasi

Populasi adalah seluruh objek, individu, kejadian, atau observasi yang ingin kita pahami. Misalnya:

- semua mahasiswa di kampus
- semua pelanggan toko
- semua transaksi bulan ini
- semua rumah di satu kota

B. Apa itu sampel

Sampel adalah sebagian dari populasi yang kita ambil untuk dianalisis. Referensi Saltz menempatkan konsep **population** dan **sampling** sebagai fondasi penting sebelum masuk ke *sampling distribution* dan keputusan statistik.

Contoh:

- dari 10.000 pelanggan, kita survei 300 orang
- dari 5.000 mahasiswa, kita ambil 120 sebagai sampel
- dari seluruh data penjualan tahunan, kita analisis 2 bulan dulu

C. Mengapa kita memakai sampel

Karena dalam dunia nyata:

- mengambil seluruh populasi sering **mahal**
- kadang **tidak mungkin**
- kadang **terlalu lama**
- kadang populasinya bahkan **masih terus berubah**

Karena itu, kita memakai sampel untuk mendapatkan gambaran yang cukup baik tentang populasi.

D. Resiko memakai sampel

Masalahnya, **sampel tidak selalu persis sama dengan populasi**. Bisa saja rata-rata sampel sedikit lebih tinggi atau lebih rendah. Inilah mengapa inferensi statistik diperlukan: untuk mengelola ketidakpastian itu.

E. Ilustrasi intuitif

Bayangkan ada satu toples berisi 1.000 kelereng dengan warna campuran. Kita ingin tahu proporsi merah, biru, dan hijau. Menghitung semua tentu bisa, tetapi jika waktunya terbatas, kita bisa mengambil 50 atau 100 sebagai sampel. Dari sampel itu kita memperkirakan isi toples secara keseluruhan.

F. Contoh source code Python lengkap

Simpan sebagai:

```
src/sampel_vs_populasi.py
```

Isi file:

```
import random
import statistics

# Populasi penuh
populasi_nilai = [
    65, 70, 72, 75, 78, 80, 82, 84, 85, 86,
    88, 90, 91, 92, 94, 95, 76, 77, 79, 81,
    83, 87, 89, 93, 96, 98, 74, 73, 68, 67
]

# Statistik populasi
mean_populasi = statistics.mean(populasi_nilai)

# Ambil sampel acak
random.seed(42)
sampel = random.sample(populasi_nilai, 10)
mean_sampel = statistics.mean(sampel)

print("=== POPULASI ===")
print(populasi_nilai)
print("Ukuran populasi:", len(populasi_nilai))
print("Mean populasi :", mean_populasi)

print("\n=== SAMPEL ===")
print(sampel)
print("Ukuran sampel:", len(sampel))
print("Mean sampel :", mean_sampel)

print("\nSelisih mean sampel terhadap populasi:", mean_sampel -
mean_populasi)
```

G. Pelajaran dari contoh ini

Kode di atas menunjukkan bahwa:

- mean sampel bisa mendekati mean populasi
- tetapi tidak selalu persis sama
- semakin baik kualitas sampel, semakin masuk akal inferensinya

11.3 Konsep probabilitas dasar

Sebelum memahami inferensi, pembaca perlu nyaman dulu dengan **probabilitas**. Referensi Malato menyebut probabilitas sebagai ukuran **ketidakpastian** atas terjadinya suatu peristiwa, dan menjelaskan pendekatan frekuensi melalui **relative frequency**: bila sebuah eksperimen diulang banyak kali, proporsi kejadian akan mendekati probabilitas sebenarnya.

A. Probabilitas sebagai ukuran ketidakpastian

Kalau kita melempar koin, hasilnya tidak pasti untuk satu lemparan, tetapi kita tahu:

- peluang kepala = 0,5
- peluang ekor = 0,5

Probabilitas membantu kita berbicara tentang ketidakpastian secara terukur.

B. Probabilitas empiris

Dalam data nyata, kita sering memperkirakan probabilitas dari data historis.

Contoh dari referensi:

- jika 1.200 dari 50.000 kunjungan situs berujung pembelian, maka probabilitas konversi kira-kira 1.200 / 50.000 = 0,024 atau 2,4%

C. Event, komplemen, dan probabilitas bersyarat

Dari referensi Malato, beberapa konsep dasar yang penting:

- **event**: kejadian yang kita minati
- **komplemen**: kejadian kebalikannya
- **conditional probability**: peluang suatu kejadian jika kejadian lain sudah diketahui terjadi

D. Kenapa probabilitas penting untuk *data science*

Karena hampir semua inferensi dibangun di atas ide berikut:

- data yang kita lihat bisa muncul karena **pola nyata**
- atau hanya karena **variasi acak**

Probabilitas memberi kita bahasa untuk membedakan keduanya.

E. Source code Python lengkap: simulasi probabilitas sederhana

Simpan sebagai:

`src/probabilitas_dasar.py`

Isi file:

```
import random

# Simulasi lempar koin
random.seed(42)

jumlah_lempar = 1000
jumlah_kepala = 0

for _ in range(jumlah_lempar):
    hasil = random.choice(["K", "E"])  # K = kepala, E = ekor
    if hasil == "K":
        jumlah_kepala += 1

prob_kepala_empiris = jumlah_kepala / jumlah_lempar

print("Jumlah lemparan :", jumlah_lempar)
print("Jumlah kepala   :", jumlah_kepala)
print("Probabilitas empiris kepala:", prob_kepala_empiris)
```

F. Intuisi yang harus ditangkap

Semakin banyak pengulangan, probabilitas empiris biasanya semakin mendekati probabilitas teoritis. Ini terkait dengan **law of large numbers**, yang juga dibahas di referensi Saltz.

11.4 Distribusi sampel, *law of large numbers*, dan *central limit theorem*

Ini mungkin terdengar besar, tetapi kita akan bahas dengan santai.

A. *Law of Large Numbers*

Referensi Saltz menjelaskan bahwa jika suatu proses statistik diulang sangat banyak kali, hasil rata-ratanya akan **cenderung konvergen ke nilai stabil**. Ini adalah inti *law of large numbers*.

Artinya:

- satu sampel bisa meleset
- banyak sampel akan lebih stabil

B. Distribusi sampel

Kalau kita mengambil banyak sampel dari populasi yang sama, lalu menghitung rata-rata tiap sampel, maka kita akan memperoleh **distribusi dari rata-rata sampel**. Saltz menyebut ini sebagai *sampling distribution*.

C. *Central Limit Theorem* secara intuitif

Referensi Saltz menekankan dua hal penting dari *central limit theorem*:

- distribusi rata-rata sampel cenderung membentuk pola mendekati normal
- pusat distribusi itu mendekati mean populasi, terutama saat ukuran sampel cukup besar

D. Mengapa ini penting

Karena banyak teknik inferensial bergantung pada ide bahwa rata-rata sampel punya perilaku yang cukup teratur, bahkan jika data aslinya tidak sepenuhnya normal.

E. Source code Python lengkap: simulasi distribusi rata-rata sampel

Simpan sebagai:

```
src/distribusi_sampling_mean.py
```

Isi file:

```
import random
import statistics
import matplotlib.pyplot as plt

random.seed(42)

# Populasi
populasi = [45, 50, 48, 52, 47, 55, 60, 43, 49, 51, 46, 54, 58, 62,
44, 53]

# Ulangi pengambilan sampel
rata_rata_sampel = []

jumlah_replikasi = 1000
ukuran_sampel = 5

for _ in range(jumlah_replikasi):
    sampel = random.choices(populasi, k=ukuran_sampel)  # sampling
with replacement
    rata_rata_sampel.append(statistics.mean(sampel))
```

```
print("Mean populasi:", statistics.mean(populasi))
print("Mean dari mean sampel:", statistics.mean(rata_rata_sampel))

plt.hist(rata_rata_sampel, bins=30)
plt.title("Distribusi Mean Sampel")
plt.xlabel("Mean Sampel")
plt.ylabel("Frekuensi")
plt.tight_layout()
plt.show()
```

F. Pelajaran dari kode ini

Pembaca akan melihat bahwa:

- satu sampel bisa berbeda-beda
- tetapi rata-rata dari banyak mean sampel akan mendekati mean populasi
- distribusinya cenderung lebih rapi dan sering mendekati bentuk lonceng

11.5 Uji sederhana secara intuitif

Ini bagian yang sering membuat pemula takut. Padahal ide dasarnya sederhana.

A. Apa itu uji hipotesis

Uji hipotesis adalah cara formal untuk bertanya:

> **apakah pola yang saya lihat cukup kuat, atau bisa saja hanya kebetulan?**

Referensi Malato menjelaskan uji satu sampel *Student's t-test* sebagai uji untuk menentukan apakah mean sampel berasal dari populasi dengan mean tertentu. Ia juga menekankan bahwa uji ini dibangun di atas dua hipotesis:

- **H0**: hipotesis nol
- **H1**: hipotesis alternatif

B. Intuisi H0 dan H1

Misalnya kita ingin menguji apakah rata-rata nilai kelas sama dengan 75.

- **H0**: rata-rata populasi = 75
- **H1**: rata-rata populasi ≠ 75

Kita lalu melihat data sampel, menghitung statistik uji, dan akhirnya menghasilkan **p-value**.

C. Apa itu p-value secara sederhana

Referensi Malato berkali-kali menegaskan bahwa keputusan umum dibuat dengan membandingkan **p-value** terhadap tingkat signifikansi, sering 0,05:

- jika `p-value < 0.05`, kita menolak H0
- jika `p-value >= 0.05`, kita tidak punya cukup bukti untuk menolak H0

D. Penting: “menolak H0” bukan berarti “membuktikan H1 mutlak”

Ini poin edukatif yang sangat penting. Statistik inferensial bekerja dengan bahasa **bukti**, bukan kepastian mutlak.

E. Contoh uji satu sampel secara intuitif

Simpan sebagai:

```
src/uji_t_satu_sampel.py
```

Isi file:

```
import numpy as np
from scipy import stats

# Data sampel nilai
nilai = np.array([78, 82, 80, 79, 81, 83, 77, 84, 80, 82])

# Hipotesis:
# H0: mean = 75
# H1: mean != 75

nilai_pembanding = 75

t_stat, p_value = stats.ttest_1samp(nilai,
popmean=nilai_pembanding)

print("Data sampel:", nilai)
print("Mean sampel:", np.mean(nilai))
print("t-statistic:", t_stat)
print("p-value    :", p_value)

alpha = 0.05

if p_value < alpha:
    print("Keputusan: Tolak H0")
    print("Interpretasi: rata-rata sampel berbeda signifikan
dari 75")
else:
    print("Keputusan: Gagal menolak H0")
    print("Interpretasi: belum ada bukti cukup bahwa rata-rata
berbeda dari 75")
```

F. Kapan uji sederhana ini berguna

Contoh pertanyaan yang cocok:

- apakah rata-rata skor kepuasan berbeda dari target
- apakah rata-rata berat produk sesuai standar
- apakah rata-rata nilai kelas tampak berbeda dari ambang tertentu

Referensi Malato memberi contoh-contoh bisnis yang sangat serupa: berat produk, pertumbuhan divisi, dan skor kepuasan pelanggan.

11.6 Contoh uji perbedaan dua kelompok secara intuitif

Selain satu sampel, kita sering ingin tahu apakah dua kelompok tampak berbeda.

Contoh:

- nilai kelas A vs kelas B
- penjualan sebelum promosi vs sesudah promosi
- kepuasan pengguna fitur lama vs fitur baru

Source code Python lengkap

Simpan sebagai:

```
src/uji_dua_sampel_sederhana.py
```

Isi file:

```
import numpy as np
from scipy import stats

# Dua kelompok data
kelas_A = np.array([78, 80, 82, 79, 81, 83, 77, 84])
kelas_B = np.array([72, 74, 70, 75, 73, 71, 76, 72])

print("Mean kelas A:", np.mean(kelas_A))
print("Mean kelas B:", np.mean(kelas_B))

# Uji t dua sampel independen
t_stat, p_value = stats.ttest_ind(kelas_A, kelas_B, equal_var=False)

print("t-statistic:", t_stat)
print("p-value    :", p_value)

alpha = 0.05

if p_value < alpha:
    print("Keputusan: Tolak H0")
    print("Interpretasi: terdapat perbedaan mean yang signifikan antara dua
kelas")
else:
    print("Keputusan: Gagal menolak H0")
```

```
    print("Interpretasi: belum ada bukti cukup bahwa mean kedua kelas
berbeda")
```

Intuisi yang harus dipahami

Kalau rata-rata dua kelompok berbeda, kita masih harus bertanya:

apakah perbedaan itu cukup besar dibanding variasi acaknya?

Itulah yang dibantu oleh uji statistik.

11.7 Mengapa statistik penting untuk interpretasi, bukan sekadar hitung

Ini bagian paling penting dari bab ini.

Sering kali pemula merasa statistik hanya kumpulan rumus:

- hitung mean
- hitung p-value
- selesai

Padahal referensi Malato justru menekankan hal yang sebaliknya: statistik harus membantu kita **menerjemahkan masalah nyata menjadi pertanyaan statistik**, memilih alat yang tepat, dan **mengkomunikasikan hasil dengan benar**. Ia bahkan menyebut keterampilan metodologis dan komunikasi ini sebagai komponen yang sangat penting bagi profesi analis data.

A. Statistik membantu interpretasi

Statistik membantu kita menjawab:

- apakah hasil ini stabil atau kebetulan
- apakah ukuran sampel cukup masuk akal
- apakah perbedaan yang terlihat cukup berarti
- apakah asumsi metode layak dipenuhi
- seberapa yakin kita pada simpulan

B. Statistik membantu membatasi ego analitis

Tanpa statistik, analis mudah berkata:

- “grafiknya naik, berarti ada efek”
- “rata-ratanya beda, berarti pasti signifikan”
- “model saya bagus, berarti hasilnya benar”

Padahal statistik mengajarkan kehati-hatian:

- belum tentu
- tergantung variabilitas
- tergantung ukuran sampel

- tergantung asumsi
- tergantung konteks

C. Statistik dan *machine learning* saling melengkapi

Referensi Lim membedakan pendekatan statistik klasik yang kuat pada **estimation, inference, testing, prediction, dan grouping** dari sebagian pendekatan *machine learning* yang sering lebih fokus pada prediksi. Ini penting: dalam *data science*, kita tidak hanya ingin model yang bisa memprediksi, tetapi juga ingin tahu **bagaimana menafsirkan hasil dengan benar**.

D. Contoh konkret

Misalnya sebuah model menunjukkan bahwa pelanggan yang menerima promosi berbelanja lebih banyak. Statistik inferensial membantu kita bertanya:

- apakah peningkatan itu benar-benar meyakinkan
- atau hanya kebetulan pada sampel kecil ini
- apakah variasi data terlalu besar
- apakah ada bukti yang cukup kuat

E. Inti pesan bab ini

Statistik penting bukan karena rumusnya, tetapi karena ia melatih cara berpikir yang jujur terhadap ketidakpastian.

11.8 Source code Python lengkap: bab terpadu

Berikut program terpadu yang menggabungkan ide utama bab ini.

Simpan sebagai:

```
src/bab11_statistik_inferensial_ringan.py
```

Isi file:

```
import random
import statistics
import numpy as np
import matplotlib.pyplot as plt
from scipy import stats

print("=" * 70)
print("BAB 11 - STATISTIK INFERENSIAL RINGAN UNTUK DATA SCIENCE")
print("=" * 70)

# ============================================================
# BAGIAN A - POPULASI VS SAMPEL
# ============================================================
populasi = [45, 50, 48, 52, 47, 55, 60, 43, 49, 51, 46, 54, 58, 62, 44, 53]
random.seed(42)
```

```
sampel = random.sample(populasi, 6)

print("\n[ A ] POPULASI VS SAMPEL")
print("Populasi       :", populasi)
print("Mean populasi  :", statistics.mean(populasi))
print("Sampel         :", sampel)
print("Mean sampel    :", statistics.mean(sampel))

# ============================================================
# BAGIAN B - SIMULASI DISTRIBUSI MEAN SAMPEL
# ============================================================
rata_rata_sampel = []
jumlah_replikasi = 1000
ukuran_sampel = 5

for _ in range(jumlah_replikasi):
    s = random.choices(populasi, k=ukuran_sampel)
    rata_rata_sampel.append(statistics.mean(s))

print("\n[ B ] DISTRIBUSI MEAN SAMPEL")
print("Mean populasi            :", statistics.mean(populasi))
print("Mean dari mean sampel    :", statistics.mean(rata_rata_sampel))

plt.figure(figsize=(8, 5))
plt.hist(rata_rata_sampel, bins=30)
plt.title("Distribusi Mean Sampel")
plt.xlabel("Mean Sampel")
plt.ylabel("Frekuensi")
plt.tight_layout()
plt.show()

# ============================================================
# BAGIAN C - UJI t SATU SAMPEL
# ============================================================
nilai = np.array([78, 82, 80, 79, 81, 83, 77, 84, 80, 82])
nilai_pembanding = 75

t_stat_1, p_val_1 = stats.ttest_1samp(nilai, popmean=nilai_pembanding)

print("\n[ C ] UJI t SATU SAMPEL")
print("Data nilai     :", nilai)
print("Mean sampel    :", np.mean(nilai))
print("Pembanding     :", nilai_pembanding)
print("t-statistic    :", t_stat_1)
print("p-value        :", p_val_1)

if p_val_1 < 0.05:
    print("Keputusan      : Tolak H0")
else:
    print("Keputusan      : Gagal menolak H0")

# ============================================================
# BAGIAN D - UJI t DUA SAMPEL
# ============================================================
kelas_A = np.array([78, 80, 82, 79, 81, 83, 77, 84])
kelas_B = np.array([72, 74, 70, 75, 73, 71, 76, 72])

t_stat_2, p_val_2 = stats.ttest_ind(kelas_A, kelas_B, equal_var=False)
```

```
print("\n[ D ] UJI t DUA SAMPEL")
print("Mean kelas A   :", np.mean(kelas_A))
print("Mean kelas B   :", np.mean(kelas_B))
print("t-statistic    :", t_stat_2)
print("p-value        :", p_val_2)

if p_val_2 < 0.05:
    print("Keputusan      : Tolak H0")
else:
    print("Keputusan      : Gagal menolak H0")

print("\nSELESAI. Fokus utama bukan hanya angka, tetapi interpretasi
hasil.")
print("=" * 70)
```

11.9 Langkah kerja yang disarankan untuk pembaca

Agar bab ini benar-benar membentuk kebiasaan berpikir yang baik, berikut urutan yang saya sarankan:

- **Langkah 1: pahami dulu populasi dan sampel**
  Jangan langsung bicara p-value kalau konsep ini belum nyaman.
- **Langkah 2: lihat probabilitas sebagai ukuran ketidakpastian**
  Biasakan berpikir dalam bahasa peluang, bukan kepastian mutlak.
- **Langkah 3: pahami distribusi mean sampel**
  Ini membuat inferensi terasa lebih masuk akal.
- **Langkah 4: baru masuk ke uji sederhana**
  Mulai dari uji satu sampel atau dua sampel.
- **Langkah 5: fokus pada interpretasi**
  Jangan hanya membaca angka hasil fungsi statistik.
- **Langkah 6: selalu hubungkan ke konteks masalah**
  Statistik yang baik selalu kembali ke pertanyaan nyata.

11.10 Kesalahan umum pemula

Ada beberapa jebakan yang sangat sering terjadi:

- **mengira sampel pasti sama dengan populasi**
- **menganggap p-value kecil sebagai bukti mutlak**
- **mengira tidak menolak H0 berarti H0 pasti benar**
- **melupakan asumsi dasar uji statistik**
- **menggunakan statistik untuk pembenaran, bukan interpretasi**
- **terlalu fokus pada rumus, lupa konteks masalah**

Kebiasaan yang lebih sehat adalah selalu bertanya:

> **apakah saya sedang membaca bukti, atau hanya membaca angka?**

11.11 Refleksi: statistik inferensial melatih kerendahan hati analitis

Bab ini seharusnya membuat pembaca menyadari satu hal besar: **dalam *data science*, kita hampir tidak pernah memiliki kepastian penuh.** Kita bekerja dengan sampel, asumsi, peluang, dan bukti yang tingkat kekuatannya berbeda-beda. Statistik inferensial mengajarkan bahwa analis yang baik bukan orang yang paling cepat menyimpulkan, tetapi orang yang paling hati-hati membedakan antara:

- pola nyata
- fluktuasi acak
- bukti kuat
- bukti yang masih lemah

Itulah mengapa statistik sangat penting untuk interpretasi. Ia membuat kita lebih jujur, lebih kritis, dan lebih matang saat membaca hasil analisis maupun model. Ini juga sejalan dengan semangat referensi yang menekankan praktik, metodologi, dan kemampuan mengkomunikasikan hasil dengan benar.

Penutup Bab

Statistik inferensial ringan adalah **fondasi penting** bagi pembaca yang ingin masuk ke *data science* dengan cara yang sehat. Melalui bab ini, pembaca telah mengenal **sampel vs populasi**, **konsep probabilitas dasar**, **sampling distribution**, **law of large numbers**, **central limit theorem**, dan **uji sederhana secara intuitif**, lalu memahami **mengapa statistik penting untuk interpretasi, bukan sekadar hitung**. Semua ini sangat sejalan dengan outline buku utama dan kuat ditopang oleh materi lampiran tentang probability, sampling, central limit theorem, hypothesis testing, dan inferential statistics.

Pesan terpenting dari bab ini sederhana: **dalam data science, angka yang terlihat meyakinkan tetap harus dibaca dengan rasa hormat pada ketidakpastian**.

Tantangan kecil untuk pembaca: modifikasi file `bab11_statistik_inferensial_ringan.py` agar program juga menghitung dan membandingkan mean dari 100, 1.000, dan 10.000 replikasi sampel, lalu lihat bagaimana hasilnya makin mendekati mean populasi.

Bab 12. Mini Project 2 — EDA dan Statistik Lengkap

Bab ini adalah **titik transisi yang sangat penting**. Setelah pembaca mempelajari Python dasar, NumPy, Pandas, pembersihan data, visualisasi, statistik deskriptif, regresi linear, dan statistik inferensial ringan, kini saatnya semua itu dipakai bersama dalam satu alur kerja yang utuh. Di outline naskah utama, mini project ini memang ditempatkan sebagai latihan **membaca dataset, melakukan cleaning, statistik deskriptif, korelasi, visualisasi, insight awal, dan menulis narasi temuan**. Pada versi penguatan outline, semangatnya tetap sama: pembaca tidak hanya "menjalankan library", tetapi benar-benar belajar **membaca data, memahami struktur masalah, lalu menyusun cerita berbasis bukti**.

Arah ini juga sangat konsisten dengan materi lampiran. Jain menempatkan **EDA** sebagai tahap vital untuk memahami struktur, pola, dan hubungan dalam dataset, dengan unsur inti berupa **descriptive statistics, correlation analysis, trend identification, dan visual summaries**, lalu dilanjutkan ke *case study* EDA pada dataset nyata. Saltz juga menekankan bahwa statistik deskriptif dipakai untuk memberi gambaran ringkas atas dataset yang terlalu besar untuk dibaca satu per satu, sedangkan materi lain menekankan bahwa proyek deskriptif yang baik harus menghasilkan **insight**, bukan hanya output teknis.

Bab ini dirancang agar pembaca mampu melakukan satu alur kerja EDA yang lengkap:

- **load dataset**
- **cleaning**
- **statistik deskriptif**
- **korelasi**
- **regresi linear sederhana**
- **insight awal**
- **menulis narasi temuan**

Supaya latihan terasa nyata tetapi tetap ramah pemula, kita akan memakai dataset sederhana bertema **kebiasaan belajar mahasiswa**. Dataset ini cukup kaya untuk latihan statistik, korelasi, dan regresi linear sederhana, namun tetap mudah dipahami secara konteks.

12.1 Tujuan mini project

Mini project ini bertujuan untuk melatih pembaca agar mampu:

- membaca dataset dari file CSV
- mengenali masalah kualitas data
- membersihkan data yang kotor
- membuat ringkasan statistik
- melihat hubungan antarvariabel
- membangun model regresi linear sederhana
- menulis insight awal dengan bahasa yang jelas
- menyusun narasi temuan yang siap masuk laporan atau notebook

Ini penting karena dalam dunia nyata, pekerjaan analis jarang berhenti di kode. Hasil akhir yang diharapkan bukan hanya *output*, tetapi juga **pemahaman** dan **narasi**. Referensi lampiran juga menekankan bahwa proyek yang baik harus terdokumentasi dengan jelas, disertai penjelasan, interpretasi, dan ringkasan temuan.

12.2 Studi kasus mini project

Skenario

Misalkan kita ingin menganalisis dataset sederhana tentang mahasiswa, dengan pertanyaan awal berikut:

- apakah **jam belajar** berkaitan dengan **nilai**?
- apakah **kehadiran** dan **tugas** juga tampak berhubungan dengan hasil belajar?
- apakah ada data yang perlu dibersihkan lebih dulu?
- variabel mana yang paling masuk akal dijadikan prediktor awal untuk nilai?

Kita tidak langsung melompat ke model yang rumit. Kita mulai dari hal-hal yang paling sehat:

1. **baca data**
2. **rapikan data**
3. **ringkas statistik**
4. **lihat korelasi**
5. **coba regresi linear sederhana**
6. **tarik insight awal**
7. **tulis narasi temuan**

12.3 Menyiapkan dataset

Simpan file berikut sebagai:

```
data/raw/eda_mahasiswa.csv
```

Isi file:

```
id,nama,jam_belajar,kehadiran,tugas,nilai,kota
1,Andi,6,82,78,80,Jakarta
2,Budi,7,85,80,84,Bandung
3,Citra,8,88,84,89,Depok
4,Dina,5,79,74,76,Jakarta
5,Eka,9,90,88,92,Bandung
6,Fajar,10,94,91,95,Depok
7,Gina,4,75,70,72,Jakarta
8,Hadi,7,86,81,85,Surabaya
9,Ika,8,89,86,90,Jakarta
10,Joko,5,78,73,75,Bandung
11,Kiki,6,83,79,81,Depok
12,Lina,11,96,94,98,Jakarta
```

```
13,Mira,6,81,77,79,Bandung
14,Nanda,7,84,80,83,
15,Oki,3,72,68,69,Depok
16,Prita,9,91,89,93,Jakarta
17,Qori,6,82,78,80,Jakarta
17,Qori,6,82,78,80,Jakarta
18,Raka,10,93,90,96,Bandung
19,Sinta,8,87,85,88,Surabaya
20,Tono,4,74,69,71,Jakarta
21,Uli,dua belas,97,95,99,Depok
22,Vina,7,,82,86,Bandung
23,Wawan,8,88,,87,Jakarta
24,Xena,12,98,96,150,Depok
```

Dataset ini sengaja dibuat **sedikit kotor**, karena mini project EDA yang baik hampir selalu dimulai dari data yang belum sempurna. Di sini kita punya beberapa masalah nyata:

- ada **nilai kota yang kosong**
- ada **baris duplikat**
- ada **jam_belajar** yang ditulis sebagai teks: `"dua belas"`
- ada **kehadiran** yang kosong
- ada **tugas** yang kosong
- ada **nilai ekstrem** pada kolom `nilai`, yaitu `150`

Masalah seperti ini sangat cocok untuk latihan *cleaning* sebelum analisis.

12.4 Langkah 1 — Load dataset

Langkah pertama adalah membaca dataset dan melihat bentuk awalnya. Praktik ini sangat konsisten dengan pola EDA di materi lampiran, yang menyarankan memulai dari `read_csv()`, `head()`, `info()`, dan ringkasan statistik awal.

Simpan file berikut sebagai:

```
src/bab12_eda_langkah1_load.py
```

Isi file:

```
import pandas as pd

df = pd.read_csv("data/raw/eda_mahasiswa.csv")

print("=== LIMA BARIS PERTAMA ===")
print(df.head())

print("\n=== SHAPE ===")
print(df.shape)

print("\n=== INFO ===")
print(df.info())
```

```
print("\n=== TIPE DATA ===")
print(df.dtypes)
```

Apa yang harus diperhatikan

Saat menjalankan kode ini, pembaca perlu melihat:

- jumlah baris dan kolom
- apakah ada kolom yang tidak terbaca sebagai numerik
- apakah ada nilai kosong
- apakah bentuk dataset sesuai harapan

Pada tahap ini, fokus kita bukan menyimpulkan, tetapi **mengenali kondisi awal data**.

12.5 Langkah 2 — Cleaning

Tahap *cleaning* adalah fondasi mini project ini. Sebelumnya di bab *data cleaning*, pembaca sudah belajar missing value, duplikat, tipe data salah, outlier, dan encoding. Di mini project ini, semuanya dipakai bersama dalam satu alur kerja.

A. Tujuan cleaning pada mini project ini

Kita akan melakukan:

- hapus duplikat
- perbaiki tipe data salah
- tangani missing value
- cek outlier dasar
- simpan data bersih

B. Source code lengkap untuk cleaning

Simpan file berikut sebagai:

```
src/bab12_eda_langkah2_cleaning.py
```

Isi file:

```
import pandas as pd

# 1. Baca data
df = pd.read_csv("data/raw/eda_mahasiswa.csv")

print("=== DATA AWAL ===")
print(df)
print("\nShape awal:", df.shape)

# 2. Hapus duplikat
```

```
df = df.drop_duplicates()
print("\n=== SETELAH HAPUS DUPLIKAT ===")
print("Shape sekarang:", df.shape)

# 3. Perbaiki tipe data
df["jam_belajar"] = pd.to_numeric(df["jam_belajar"], errors="coerce")
df["kehadiran"] = pd.to_numeric(df["kehadiran"], errors="coerce")
df["tugas"] = pd.to_numeric(df["tugas"], errors="coerce")
df["nilai"] = pd.to_numeric(df["nilai"], errors="coerce")

print("\n=== TIPE DATA SETELAH KONVERSI ===")
print(df.dtypes)

# 4. Cek missing value
print("\n=== JUMLAH MISSING VALUE ===")
print(df.isnull().sum())

# 5. Isi missing numerik dengan median
for kolom in ["jam_belajar", "kehadiran", "tugas", "nilai"]:
    df[kolom] = df[kolom].fillna(df[kolom].median())

# 6. Isi missing kategorikal dengan mode
df["kota"] = df["kota"].fillna(df["kota"].mode()[0])

print("\n=== SETELAH MENANGANI MISSING VALUE ===")
print(df)

# 7. Deteksi outlier sederhana dengan IQR untuk kolom nilai
Q1 = df["nilai"].quantile(0.25)
Q3 = df["nilai"].quantile(0.75)
IQR = Q3 - Q1
batas_bawah = Q1 - 1.5 * IQR
batas_atas = Q3 + 1.5 * IQR

print("\n=== BATAS OUTLIER NILAI ===")
print("Q1          :", Q1)
print("Q3          :", Q3)
print("IQR         :", IQR)
print("Batas bawah :", batas_bawah)
print("Batas atas  :", batas_atas)

outlier = df[(df["nilai"] < batas_bawah) | (df["nilai"] > batas_atas)]
print("\n=== DATA OUTLIER ===")
print(outlier)

# 8. Tangani outlier dengan winsor sederhana: ganti nilai di atas
batas atas menjadi median
median_nilai = df["nilai"].median()
df.loc[df["nilai"] > batas_atas, "nilai"] = median_nilai
df.loc[df["nilai"] < batas_bawah, "nilai"] = median_nilai

print("\n=== SETELAH MENANGANI OUTLIER ===")
print(df)
```

```
# 9. Simpan hasil cleaning
df.to_csv("outputs/eda_mahasiswa_bersih.csv", index=False)
print("\nFile data bersih disimpan di:
outputs/eda_mahasiswa_bersih.csv")
```

Apa yang dilatih di tahap ini

Pembaca belajar bahwa EDA yang baik selalu dimulai dengan pertanyaan seperti:

- apakah data ini sudah layak dianalisis?
- apakah ada baris ganda?
- apakah angka masih terbaca sebagai teks?
- apakah ada nilai ekstrim yang bisa merusak ringkasan atau model?

Itulah sebabnya *cleaning* tidak boleh diperlakukan sebagai formalitas.

12.6 Langkah 3 — Statistik deskriptif

Setelah data dibersihkan, kita bisa mulai merangkum data secara statistik. Ini sejalan dengan materi lampiran yang menempatkan **summary statistics** sebagai inti *descriptive analytics*: mean, median, standard deviation, dan ukuran dispersi lain membantu memberi gambaran cepat tentang data.

Source code lengkap

Simpan sebagai:

```
src/bab12_eda_langkah3_statistik.py
```

Isi file:

```
import pandas as pd

df = pd.read_csv("outputs/eda_mahasiswa_bersih.csv")

print("=== DESCRIBE ===")
print(df[["jam_belajar", "kehadiran", "tugas",
"nilai"]].describe())

print("\n=== MEAN, MEDIAN, STD ===")
for kolom in ["jam_belajar", "kehadiran", "tugas", "nilai"]:
    print("=" * 50)
    print("Kolom  :", kolom)
    print("Mean   :", df[kolom].mean())
    print("Median :", df[kolom].median())
    print("Std    :", df[kolom].std())
    print("Min    :", df[kolom].min())
    print("Max    :", df[kolom].max())
```

Cara membaca hasilnya

Saat membaca hasil statistik deskriptif, perhatikan:

- **mean** dan **median**: apakah nilainya dekat atau jauh
- **std**: apakah variasi datanya besar
- **min** dan **max**: apakah rentangnya masuk akal
- apakah masih ada indikasi nilai ekstrim

Tujuan tahap ini

Bukan sekadar menghasilkan angka, tetapi menjawab:

- bagaimana pusat data?
- bagaimana sebarannya?
- apakah nilai mahasiswa relatif homogen atau bervariasi?

12.7 Langkah 4 — Korelasi

Setelah statistik deskriptif, langkah berikutnya adalah melihat hubungan antar variabel. Materi lampiran menegaskan bahwa **correlation analysis** adalah komponen inti EDA, terutama untuk memahami hubungan linear maupun berbasis peringkat antar variabel.

Pertanyaan analitis

Dalam mini project ini, kita tertarik pada pertanyaan seperti:

- apakah **jam_belajar** berkorelasi dengan **nilai**
- apakah **kehadiran** juga berkorelasi
- apakah **tugas** tampak memiliki hubungan yang kuat

Source code lengkap

Simpan sebagai:

```
src/bab12_eda_langkah4_korelasi.py
```

Isi file:

```
import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt

df = pd.read_csv("outputs/eda_mahasiswa_bersih.csv")

kolom_numerik = ["jam_belajar", "kehadiran", "tugas", "nilai"]
corr_pearson = df[kolom_numerik].corr(method="pearson")
corr_spearman = df[kolom_numerik].corr(method="spearman")

print("=== KORELASI PEARSON ===")
```

```
print(corr_pearson)

print("\n=== KORELASI SPEARMAN ===")
print(corr_spearman)

plt.figure(figsize=(8, 6))
sns.heatmap(corr_pearson, annot=True, cmap="coolwarm")
plt.title("Heatmap Korelasi Pearson")
plt.tight_layout()
plt.show()
```

Cara membaca korelasi

Fokus utama:

- nilai mendekati **1** → hubungan positif kuat
- nilai mendekati **0** → hubungan lemah
- nilai mendekati **-1** → hubungan negatif kuat

Catatan penting

Korelasi **tidak sama** dengan sebab-akibat. Tujuannya di sini adalah memberi sinyal awal:

- variabel mana yang tampak relevan
- apakah hubungan cukup kuat untuk dijadikan kandidat prediktor
- apakah ada fitur yang terlalu mirip satu sama lain

12.8 Langkah 5 — Regresi linear sederhana

Setelah melihat korelasi, kita bisa mencoba model sederhana: **regresi linear sederhana**. Ini sangat sejalan dengan struktur outline yang menjadikan regresi linear sebagai jembatan dari statistik ke machine learning, dan mini project ini menjadi tempat ideal untuk menerapkannya secara utuh.

Pertanyaan model

Kita akan mulai dari pertanyaan sederhana:

bisakah jam_belajar dipakai untuk memprediksi nilai?

Source code lengkap

Simpan sebagai:

`src/bab12_eda_langkah5_regresi.py`

Isi file:

```
import pandas as pd
import matplotlib.pyplot as plt
from sklearn.linear_model import LinearRegression
from sklearn.metrics import mean_absolute_error,
mean_squared_error, r2_score
import math

# 1. Baca data bersih
df = pd.read_csv("outputs/eda_mahasiswa_bersih.csv")

# 2. Pilih fitur dan target
X = df[["jam_belajar"]]
y = df["nilai"]

# 3. Buat dan latih model
model = LinearRegression()
model.fit(X, y)

# 4. Prediksi
y_pred = model.predict(X)

# 5. Evaluasi
mae = mean_absolute_error(y, y_pred)
mse = mean_squared_error(y, y_pred)
rmse = math.sqrt(mse)
r2 = r2_score(y, y_pred)

print("=== PARAMETER MODEL ===")
print("Intercept :", model.intercept_)
print("Koefisien :", model.coef_[0])

print("\n=== EVALUASI MODEL ===")
print("MAE  :", mae)
print("MSE  :", mse)
print("RMSE :", rmse)
print("R²   :", r2)

# 6. Tambahkan ke dataframe
df["prediksi_nilai"] = y_pred
df["residual"] = df["nilai"] - df["prediksi_nilai"]

print("\n=== DATA DENGAN PREDIKSI ===")
print(df[["nama", "jam_belajar", "nilai", "prediksi_nilai",
"residual"]])

# 7. Visualisasi garis regresi
plt.figure(figsize=(8, 5))
plt.scatter(df["jam_belajar"], df["nilai"], label="Data aktual")
plt.plot(df["jam_belajar"], y_pred, label="Garis regresi")
plt.title("Regresi Linear Sederhana: Jam Belajar vs Nilai")
plt.xlabel("Jam Belajar")
```

```
plt.ylabel("Nilai")
plt.legend()
plt.grid(True)
plt.tight_layout()
plt.show()
```

Apa yang dipelajari di tahap ini

Pembaca belajar bahwa setelah EDA awal, kita bisa mencoba model sederhana untuk melihat apakah pola yang tampak secara statistik juga cukup kuat untuk prediksi dasar.

12.9 Langkah 6 — Insight awal

Inilah bagian yang paling penting dalam mini project: **menerjemahkan hasil menjadi insight**.

Kita tidak berhenti pada:

- `describe()`
- matriks korelasi
- `coef_`
- `R²`

Sebaliknya, kita bertanya:

- apa cerita yang muncul dari data?
- faktor mana yang paling tampak relevan?
- apakah hasil model cukup masuk akal?
- adakah hal yang perlu diuji lebih lanjut?

Contoh insight awal yang mungkin muncul

Berdasarkan alur analisis di atas, pembaca sangat mungkin menemukan temuan awal seperti:

- **jam belajar** tampak memiliki hubungan positif dengan **nilai**
- **kehadiran** dan **tugas** juga kemungkinan punya hubungan positif
- setelah cleaning, data menjadi lebih stabil dan lebih layak dibaca
- ada satu nilai ekstrim yang sebelumnya berpotensi mengganggu analisis
- regresi linear sederhana memberi gambaran awal yang cukup baik, tetapi masih belum menangkap seluruh faktor yang mempengaruhi nilai

Hal yang perlu dijaga

Insight awal adalah **observasi berbasis data**, bukan klaim absolut. Bahasa yang baik misalnya:

- “terdapat indikasi”
- “tampak berkorelasi”
- “memberi sinyal awal”
- “layak diuji lebih lanjut”

Bahasa seperti ini jauh lebih sehat daripada mengatakan:

- “pasti menyebabkan”
- “terbukti secara mutlak”
- “satu-satunya faktor”

12.10 Langkah 7 — Menulis narasi temuan

Ini bagian yang sering dilupakan pemula, padahal justru sangat penting. Materi lampiran menekankan bahwa proyek analitik yang baik harus menghasilkan **interpretasi** dan **narasi**, bukan hanya tabel dan grafik. Bahkan ada penekanan bahwa notebook dan laporan harus memuat penjelasan langkah, hasil, dan insight secara jelas.

Struktur narasi temuan yang baik

Narasi temuan sebaiknya memuat:

- tujuan analisis
- sumber atau konteks dataset
- langkah pembersihan data
- ringkasan statistik utama
- hubungan antarfaktor
- hasil regresi sederhana
- insight awal
- batasan analisis

Contoh narasi temuan

Berikut contoh narasi yang bisa dimasukkan ke notebook atau laporan:

> Analisis awal terhadap dataset mahasiswa menunjukkan bahwa data mentah memerlukan pembersihan sebelum dipakai lebih lanjut. Ditemukan beberapa masalah kualitas data berupa baris duplikat, nilai hilang, tipe data yang tidak konsisten, serta satu nilai ekstrim pada kolom nilai. Setelah proses *cleaning*, dataset menjadi lebih stabil untuk dianalisis.
>
> Statistik deskriptif menunjukkan bahwa nilai mahasiswa berada pada rentang yang cukup wajar setelah pembersihan, dengan pusat data berada di sekitar nilai menengah-atas. Variabel jam belajar, kehadiran, dan tugas juga menunjukkan pola yang relatif konsisten.
>
> Analisis korelasi memberikan indikasi bahwa jam belajar, kehadiran, dan tugas memiliki hubungan positif terhadap nilai. Dari ketiga variabel tersebut, jam belajar dipilih sebagai prediktor awal untuk regresi linear sederhana karena secara intuitif mudah dipahami dan secara statistik menunjukkan hubungan yang masuk akal dengan nilai.
>
> Model regresi linear sederhana menunjukkan bahwa peningkatan jam belajar cenderung diikuti peningkatan nilai. Walaupun model ini belum menjelaskan seluruh variasi nilai mahasiswa, hasilnya cukup baik sebagai baseline awal. Dengan demikian, analisis ini memberi sinyal bahwa kebiasaan belajar memiliki peran penting, namun kemungkinan masih ada faktor lain yang juga perlu dipertimbangkan pada model yang lebih lengkap, seperti kualitas tugas, kehadiran, atau faktor motivasi belajar.

Kenapa narasi penting

Karena pada akhirnya, hasil EDA harus bisa dipahami oleh:

- dosen
- pembimbing
- atasan
- rekan tim
- pembaca laporan

Kita tidak menulis hanya untuk Python, tetapi juga untuk manusia.

12.11 Source code Python lengkap: mini project terpadu

Berikut program lengkap yang menggabungkan seluruh alur mini project ini: **load dataset, cleaning, statistik deskriptif, korelasi, regresi linear sederhana, dan ringkasan insight**.

Simpan sebagai:

```
src/bab12_mini_project_eda_lengkap.py
```

Isi file:

```
import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt
from sklearn.linear_model import LinearRegression
from sklearn.metrics import mean_absolute_error, mean_squared_error,
r2_score
import math

print("=" * 80)
print("BAB 12 - MINI PROJECT 2: EDA DAN STATISTIK LENGKAP")
print("=" * 80)

# ============================================================
# 1. LOAD DATASET
# ============================================================
df = pd.read_csv("data/raw/eda_mahasiswa.csv")

print("\n[1] DATA AWAL")
print(df.head())
print("Shape awal:", df.shape)

# ============================================================
# 2. CLEANING
# ============================================================
print("\n[2] CLEANING")

# Hapus duplikat
df = df.drop_duplicates()

# Konversi tipe data
for kolom in ["jam_belajar", "kehadiran", "tugas", "nilai"]:
    df[kolom] = pd.to_numeric(df[kolom], errors="coerce")

# Missing value
for kolom in ["jam_belajar", "kehadiran", "tugas", "nilai"]:
    df[kolom] = df[kolom].fillna(df[kolom].median())

df["kota"] = df["kota"].fillna(df["kota"].mode()[0])

# Outlier pada nilai dengan IQR
Q1 = df["nilai"].quantile(0.25)
Q3 = df["nilai"].quantile(0.75)
IQR = Q3 - Q1
batas_bawah = Q1 - 1.5 * IQR
batas_atas = Q3 + 1.5 * IQR
median_nilai = df["nilai"].median()

df.loc[df["nilai"] > batas_atas, "nilai"] = median_nilai
df.loc[df["nilai"] < batas_bawah, "nilai"] = median_nilai

print("Shape setelah cleaning:", df.shape)
print("\nMissing value setelah cleaning:")
print(df.isnull().sum())

# Simpan data bersih
```

```
df.to_csv("outputs/eda_mahasiswa_bersih.csv", index=False)

# ============================================================
# 3. STATISTIK DESKRIPTIF
# ============================================================
print("\n[3] STATISTIK DESKRIPTIF")
print(df[["jam_belajar", "kehadiran", "tugas", "nilai"]].describe())

# ============================================================
# 4. KORELASI
# ============================================================
print("\n[4] KORELASI")
kolom_numerik = ["jam_belajar", "kehadiran", "tugas", "nilai"]
corr_matrix = df[kolom_numerik].corr(method="pearson")
print(corr_matrix)

plt.figure(figsize=(8, 6))
sns.heatmap(corr_matrix, annot=True, cmap="coolwarm")
plt.title("Heatmap Korelasi Pearson")
plt.tight_layout()
plt.show()

# ============================================================
# 5. REGRESI LINEAR SEDERHANA
# ============================================================
print("\n[5] REGRESI LINEAR SEDERHANA")

X = df[["jam_belajar"]]
y = df["nilai"]

model = LinearRegression()
model.fit(X, y)

y_pred = model.predict(X)

mae = mean_absolute_error(y, y_pred)
mse = mean_squared_error(y, y_pred)
rmse = math.sqrt(mse)
r2 = r2_score(y, y_pred)

print("Intercept :", model.intercept_)
print("Koefisien :", model.coef_[0])
print("MAE       :", mae)
print("MSE       :", mse)
print("RMSE      :", rmse)
print("R²        :", r2)

df["prediksi_nilai"] = y_pred
df["residual"] = df["nilai"] - df["prediksi_nilai"]

print("\nContoh hasil prediksi:")
print(df[["nama", "jam_belajar", "nilai", "prediksi_nilai",
"residual"]].head())

plt.figure(figsize=(8, 5))
plt.scatter(df["jam_belajar"], df["nilai"], label="Data aktual")
plt.plot(df["jam_belajar"], y_pred, label="Garis regresi")
plt.title("Regresi Linear Sederhana: Jam Belajar vs Nilai")
plt.xlabel("Jam Belajar")
```

```
plt.ylabel("Nilai")
plt.legend()
plt.grid(True)
plt.tight_layout()
plt.show()

# ============================================================
# 6. INSIGHT AWAL
# ============================================================
print("\n[6] INSIGHT AWAL")
print("- Data awal memerlukan cleaning karena ada missing value, duplikat,
tipe data salah, dan outlier.")
print("- Setelah cleaning, dataset lebih layak untuk analisis statistik.")
print("- Korelasi menunjukkan hubungan positif antara jam belajar,
kehadiran, tugas, dan nilai.")
print("- Regresi linear sederhana menunjukkan bahwa jam belajar memiliki
hubungan positif terhadap nilai.")
print("- Model ini cukup baik sebagai baseline awal, tetapi belum menangkap
seluruh faktor yang memengaruhi nilai.")

print("\nFile data bersih disimpan di outputs/eda_mahasiswa_bersih.csv")
print("=" * 80)
```

12.12 Alur kerja yang sebaiknya dibiasakan

Mini project ini bisa menjadi pola kerja berulang untuk dataset lain. Saat nanti pembaca mendapat dataset Titanic, Housing, penjualan ritel, pendidikan, atau data UMKM, alur yang sehat tetap sama:

- **mulai dari load dataset**
- **cek struktur awal**
- **bersihkan data**
- **ringkas statistik**
- **lihat korelasi**
- **uji model sederhana**
- **tarik insight awal**
- **tulis narasi temuan**

Ini juga selaras dengan pendekatan EDA di materi lampiran yang menekankan bahwa analisis yang baik bergerak dari **descriptive statistics → correlation analysis → identifying trends and patterns → deriving insights and visual summaries**.

12.13 Kesalahan umum pemula

Ada beberapa jebakan yang sangat sering muncul saat mengerjakan mini project EDA:

- **langsung modeling tanpa cleaning**
- **menganggap `describe()` sudah cukup sebagai insight**
- **melihat korelasi lalu menyimpulkan sebab-akibat**
- **tidak menulis narasi hasil**
- **menghapus outlier tanpa alasan yang jelas**
- **menganggap satu model sederhana sudah cukup final**

- **tidak mendokumentasikan langkah cleaning**

Kebiasaan yang lebih baik adalah selalu bertanya:

> **apa cerita utama dari data ini, dan bukti apa yang mendukungnya?**

12.14 Refleksi: dari kode ke cara berpikir analitis

Mini project ini penting karena pembaca mulai mengalami satu hal yang sangat khas di dunia *data science*: **proyek analitik tidak berjalan lurus dari data ke model**, tetapi melalui tahap memahami, membersihkan, merangkum, menguji, dan menulis. Itulah sebabnya EDA tidak boleh dianggap sekadar langkah pembuka yang cepat selesai.

Di tahap ini, pembaca mulai berubah dari orang yang "bisa menjalankan Pandas" menjadi orang yang **mulai berpikir seperti analis**:

- tidak langsung percaya data mentah
- terbiasa memeriksa kualitas data
- melihat hubungan antar variabel
- memakai model sederhana sebagai *baseline*
- menuliskan hasil dengan bahasa yang bisa dipahami manusia

Inilah inti kekuatan mini project seperti ini.

Penutup Bab

Mini Project 2 ini menyatukan banyak fondasi penting yang sudah dipelajari sebelumnya: **load dataset, cleaning, statistik deskriptif, korelasi, regresi linear sederhana, insight awal, dan penulisan narasi temuan**. Struktur ini sangat sejalan dengan outline naskah utama yang menempatkan EDA lengkap pada dataset nyata sebagai latihan penting sebelum pembaca melangkah lebih jauh ke *machine learning* inti. Materi lampiran juga sangat mendukung pendekatan ini, terutama yang menekankan **EDA sebagai langkah penting untuk memahami struktur, pola, hubungan, dan insight** dalam data nyata.

Pesan terpenting dari bab ini sederhana: **EDA yang baik bukan sekadar membuka dataset, tetapi membangun pemahaman yang cukup kuat sebelum kita percaya pada model**.

Tantangan kecil untuk pembaca: ubah mini project ini agar selain `jam_belajar`, Anda juga mencoba `tugas` sebagai prediktor regresi linear sederhana, lalu bandingkan nilai **R^2** dan **RMSE** keduanya.

Bagian III — Synthetic Data, Feature, dan Persiapan Modeling

Bab 13. Men-generate Synthetic Data dengan Python

Bab ini sangat penting karena memperkenalkan satu keterampilan praktis yang sering sangat membantu dalam pembelajaran *machine learning*: **membuat data buatan secara sadar dan terkontrol**. Bab ini sangat cocok ditempatkan setelah fondasi Python, statistik, EDA, dan regresi, karena pembaca mulai perlu memahami bahwa tidak semua latihan harus menunggu dataset dunia nyata yang rapi. *Synthetic data* memberi cara cepat untuk **belajar pola, menguji kode, memahami perilaku algoritma, dan membuat simulasi** sebelum masuk ke data nyata yang lebih berantakan. Arah ini juga sejalan dengan outline utama buku yang menekankan jalur belajar bertahap, praktis, dan kuat di sisi eksperimen.

Materi lampiran juga mendukung pendekatan ini dari beberapa sisi. Dokumen-dokumen Python dan AI menekankan pentingnya **NumPy sebagai fondasi komputasi numerik**, **scikit-learn sebagai toolkit utama untuk regresi, klasifikasi, dan clustering**, serta pentingnya **preprocessing, noise handling, dan evaluasi model** sebelum masuk ke model yang lebih serius. Referensi lain juga menunjukkan bahwa *synthetic* atau *artificial data* dapat dipakai untuk memperkaya skenario pelatihan, menguji performa model, dan menyiapkan data saat data nyata terbatas atau belum tersedia.

Bab ini dirancang agar pembaca mampu:

- memahami **apa itu synthetic data** dan kapan dipakai
- membuat **data numerik buatan dengan NumPy**
- membuat dataset regresi dengan `make_regression`
- membuat dataset klasifikasi dengan `make_classification`
- membuat data clustering dengan `make_blobs`
- menambahkan **noise** untuk mensimulasikan kondisi nyata
- memahami **kelebihan dan keterbatasan synthetic data**

13.1 Apa itu *synthetic data* dan kapan dipakai

Secara sederhana, **synthetic data** adalah data buatan yang dibuat dengan aturan, distribusi, atau simulasi tertentu agar meniru pola data nyata. Data ini bukan hasil observasi langsung dari dunia nyata, tetapi **dibangun secara artifisial** untuk tujuan latihan, eksperimen, pengujian, atau simulasi.

Contoh sederhananya:

- kita membuat data **jam_belajar** dan **nilai** agar punya hubungan linear
- kita membuat dua kelas data pelanggan untuk latihan klasifikasi
- kita membuat tiga gugus titik agar bisa melihat cara kerja clustering
- kita menambahkan gangguan acak agar data tidak terlalu “sempurna”

Kapan synthetic data dipakai

Synthetic data sangat berguna ketika:

- **dataset nyata belum tersedia**
- data nyata terlalu sedikit
- data nyata mengandung isu privasi
- kita ingin memahami algoritma dengan pola yang lebih terkendali
- kita ingin menguji kode sebelum dipakai ke data sesungguhnya
- kita ingin membuat latihan yang mudah direproduksi

Dalam praktik pembelajaran, ini sangat membantu. Pembaca bisa fokus memahami:

- bentuk hubungan data
- bagaimana model bereaksi
- bagaimana noise memengaruhi hasil
- bagaimana visualisasi berubah saat pola data berubah

Mengapa penting untuk pemula

Bagi pemula, data nyata sering terlalu cepat menjadi kompleks:

- ada missing value
- ada duplikat
- ada outlier
- ada kolom yang tidak relevan
- ada pola yang sulit dibaca

Synthetic data memungkinkan kita mulai dari dunia yang lebih bersih, lalu **perlahan menambahkan kompleksitas**. Ini sangat bagus untuk pembelajaran bertahap.

13.2 Menyiapkan lingkungan kerja

Pastikan di Ubuntu 26.04 *virtual environment* Anda sudah aktif, lalu pasang *library* yang dibutuhkan:

```
pip install numpy pandas matplotlib scikit-learn
```

Kita akan banyak memakai:

- **NumPy** untuk membangun array numerik dan noise
- **Pandas** untuk melihat data sebagai tabel
- **Matplotlib** untuk visualisasi
- **scikit-learn** untuk generator dataset seperti `make_regression`, `make_classification`, dan `make_blobs`

Ini sejalan dengan lampiran yang menempatkan NumPy, pandas, dan scikit-learn sebagai fondasi ekosistem Python untuk AI dan *machine learning*.

13.3 Membuat data numerik buatan dengan NumPy

Langkah paling dasar dalam membuat *synthetic data* adalah memakai **NumPy**. Ini memberi kontrol penuh atas bentuk data, distribusi, rentang, dan noise.

A. Membuat data numerik sederhana

Contoh berikut membuat 20 data jam belajar secara acak antara 1 sampai 10 jam.

Simpan sebagai:

```
src/synthetic_numpy_dasar.py
```

Isi file:

```
import numpy as np
import pandas as pd

np.random.seed(42)

# Membuat data jam belajar acak
jam_belajar = np.random.randint(1, 11, size=20)

# Membuat data nilai dengan pola sederhana
nilai = 50 + (jam_belajar * 4)

df = pd.DataFrame({
    "jam_belajar": jam_belajar,
    "nilai": nilai
})

print(df)
```

B. Menambahkan variasi kecil

Kalau data terlalu sempurna, model akan terlalu mudah. Kita bisa menambahkan gangguan kecil.

```
import numpy as np
import pandas as pd

np.random.seed(42)

jam_belajar = np.random.randint(1, 11, size=20)
noise = np.random.randint(-5, 6, size=20)

nilai = 50 + (jam_belajar * 4) + noise

df = pd.DataFrame({
    "jam_belajar": jam_belajar,
    "nilai": nilai
})
```

```
print(df)
```

C. Visualisasi hasil

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

np.random.seed(42)

jam_belajar = np.random.randint(1, 11, size=20)
noise = np.random.randint(-5, 6, size=20)
nilai = 50 + (jam_belajar * 4) + noise

df = pd.DataFrame({
    "jam_belajar": jam_belajar,
    "nilai": nilai
})

plt.figure(figsize=(8, 5))
plt.scatter(df["jam_belajar"], df["nilai"])
plt.title("Synthetic Data dengan NumPy")
plt.xlabel("Jam Belajar")
plt.ylabel("Nilai")
plt.tight_layout()
plt.show()
```

Apa yang dipelajari dari langkah ini

Dengan NumPy, pembaca belajar bahwa data buatan bisa dibentuk secara eksplisit:

- kita tentukan sendiri fitur
- kita tentukan sendiri hubungan antar-variabel
- kita tentukan sendiri besarnya noise

Ini sangat baik untuk membangun intuisi awal.

13.4 Membuat dataset regresi dengan `make_regression`

Untuk kasus regresi, scikit-learn menyediakan generator yang sangat praktis, yaitu **`make_regression`**. Ini sangat cocok untuk latihan model yang memprediksi nilai numerik.

Apa yang dihasilkan

`make_regression` menghasilkan:

- **X**: fitur input
- **y**: target numerik

Kita bisa mengatur:

- jumlah sampel
- jumlah fitur
- jumlah fitur informatif
- besar noise
- *random state*

Source code lengkap

Simpan sebagai:

```
src/synthetic_make_regression.py
```

Isi file:

```
import pandas as pd
import matplotlib.pyplot as plt
from sklearn.datasets import make_regression

# Membuat dataset regresi sintetis
X, y = make_regression(
    n_samples=100,
    n_features=1,
    n_informative=1,
    noise=10,
    random_state=42
)

# Ubah menjadi DataFrame agar mudah dibaca
df = pd.DataFrame({
    "fitur_1": X[:, 0],
    "target": y
})

print(df.head())

# Visualisasi
plt.figure(figsize=(8, 5))
plt.scatter(df["fitur_1"], df["target"])
plt.title("Synthetic Regression Data")
plt.xlabel("Fitur 1")
plt.ylabel("Target")
plt.tight_layout()
plt.show()
```

Menambah fitur lebih banyak

```
import pandas as pd
from sklearn.datasets import make_regression

X, y = make_regression(
    n_samples=120,
    n_features=3,
    n_informative=2,
    noise=15,
    random_state=42
)

df = pd.DataFrame(X, columns=["fitur_1", "fitur_2", "fitur_3"])
df["target"] = y

print(df.head())
```

Apa yang bisa dipelajari

Dengan `make_regression`, pembaca bisa belajar:

- bagaimana dataset regresi dibentuk
- bagaimana noise memengaruhi target
- bagaimana satu atau beberapa fitur memengaruhi output

Ini sangat cocok untuk latihan regresi linear dan evaluasi model.

13.5 Membuat dataset klasifikasi dengan `make_classification`

Untuk kasus klasifikasi, scikit-learn menyediakan **`make_classification`**. Generator ini dipakai untuk membuat data dengan label kelas.

Cocok untuk latihan seperti:

- lulus / tidak lulus
- fraud / tidak fraud
- spam / tidak spam
- sakit / tidak sakit

Source code lengkap

Simpan sebagai:

```
src/synthetic_make_classification.py
```

Isi file:

```
import pandas as pd
import matplotlib.pyplot as plt
from sklearn.datasets import make_classification

# Membuat dataset klasifikasi sintetis
X, y = make_classification(
    n_samples=200,
    n_features=2,
    n_informative=2,
    n_redundant=0,
    n_classes=2,
    random_state=42
)

df = pd.DataFrame(X, columns=["fitur_1", "fitur_2"])
df["kelas"] = y

print(df.head())

# Visualisasi
plt.figure(figsize=(8, 5))
plt.scatter(df["fitur_1"], df["fitur_2"], c=df["kelas"])
plt.title("Synthetic Classification Data")
plt.xlabel("Fitur 1")
plt.ylabel("Fitur 2")
plt.tight_layout()
plt.show()
```

Versi tiga kelas

```
import pandas as pd
from sklearn.datasets import make_classification

X, y = make_classification(
    n_samples=300,
    n_features=4,
    n_informative=3,
    n_redundant=0,
    n_classes=3,
    n_clusters_per_class=1,
    random_state=42
)

df = pd.DataFrame(X, columns=["f1", "f2", "f3", "f4"])
df["kelas"] = y

print(df.head())
print(df["kelas"].value_counts())
```

Apa yang dipelajari

Dengan `make_classification`, pembaca bisa:

- melatih model klasifikasi tanpa harus mencari dataset dulu
- memahami pemisahan kelas
- melihat bagaimana fitur informatif membentuk batas keputusan

Referensi lampiran juga menekankan bahwa scikit-learn mendukung *supervised learning* untuk klasifikasi dan regresi melalui antarmuka yang konsisten, sehingga generator seperti ini sangat cocok untuk latihan awal.

13.6 Membuat data clustering dengan `make_blobs`

Untuk latihan *unsupervised learning*, khususnya clustering, generator yang sangat berguna adalah **`make_blobs`**. Ini membuat data berbentuk gugus atau klaster.

Kapan dipakai

Sangat cocok untuk belajar:

- K-Means
- segmentasi pelanggan
- visualisasi kelompok data
- memahami konsep centroid

Source code lengkap

Simpan sebagai:

```
src/synthetic_make_blobs.py
```

Isi file:

```
import pandas as pd
import matplotlib.pyplot as plt
from sklearn.datasets import make_blobs

# Membuat data clustering sintetis
X, y = make_blobs(
    n_samples=300,
    centers=3,
    n_features=2,
    cluster_std=1.2,
    random_state=42
)

df = pd.DataFrame(X, columns=["x1", "x2"])
df["cluster_asli"] = y
```

```
print(df.head())

plt.figure(figsize=(8, 5))
plt.scatter(df["x1"], df["x2"], c=df["cluster_asli"])
plt.title("Synthetic Clustering Data dengan make_blobs")
plt.xlabel("x1")
plt.ylabel("x2")
plt.tight_layout()
plt.show()
```

Mengubah tingkat sebaran cluster

```
from sklearn.datasets import make_blobs
import matplotlib.pyplot as plt

X, y = make_blobs(
    n_samples=300,
    centers=4,
    n_features=2,
    cluster_std=2.5,
    random_state=42
)

plt.figure(figsize=(8, 5))
plt.scatter(X[:, 0], X[:, 1], c=y)
plt.title("Cluster dengan Sebaran Lebih Besar")
plt.tight_layout()
plt.show()
```

Apa yang dipelajari

Pembaca bisa melihat bahwa:

- semakin kecil `cluster_std`, cluster makin rapat
- semakin besar `cluster_std`, cluster makin tumpang tindih
- data clustering tidak selalu punya label di dunia nyata, tetapi di data sintetis kita bisa tahu “jawaban asli”-nya untuk latihan

13.7 Menambah *noise* untuk simulasi kasus nyata

Data sintetis yang terlalu rapi bisa menipu. Model akan terlihat terlalu hebat. Karena itu, penting sekali belajar menambahkan **noise**.

Mengapa noise penting

Di dunia nyata, data hampir selalu punya:

- kesalahan pengukuran
- variasi alami
- pengaruh faktor lain yang tidak kita ukur
- fluktuasi acak

Lampiran juga menekankan pentingnya menangani *noise* dan preprocessing agar model lebih realistis dan tidak hanya bekerja pada data yang “terlalu bersih”.

A. Menambah noise pada data regresi NumPy

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

np.random.seed(42)

x = np.linspace(0, 10, 100)
y_tanpa_noise = 3 * x + 10
noise = np.random.normal(0, 5, size=100)
y_dengan_noise = y_tanpa_noise + noise

plt.figure(figsize=(8, 5))
plt.scatter(x, y_dengan_noise, label="Dengan noise")
plt.plot(x, y_tanpa_noise, label="Pola ideal")
plt.title("Simulasi Noise pada Data Regresi")
plt.xlabel("x")
plt.ylabel("y")
plt.legend()
plt.tight_layout()
plt.show()
```

B. Membandingkan noise kecil dan besar

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

np.random.seed(42)

x = np.linspace(0, 10, 100)
y_asli = 4 * x + 20

noise_kecil = np.random.normal(0, 2, size=100)
noise_besar = np.random.normal(0, 10, size=100)

y_kecil = y_asli + noise_kecil
y_besar = y_asli + noise_besar

plt.figure(figsize=(8, 5))
plt.scatter(x, y_kecil, label="Noise kecil")
plt.scatter(x, y_besar, label="Noise besar")
plt.plot(x, y_asli, label="Pola ideal")
plt.title("Perbandingan Tingkat Noise")
plt.xlabel("x")
plt.ylabel("y")
plt.legend()
plt.tight_layout()
plt.show()
```

Pelajaran penting

- noise kecil → pola masih jelas
- noise besar → hubungan lebih sulit dipelajari model
- model yang bagus harus siap menghadapi data yang tidak sempurna

13.8 Source code Python lengkap: latihan terpadu synthetic data

Simpan sebagai:

```
src/bab13_synthetic_data_terpadu.py
```

Isi file:

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
from sklearn.datasets import make_regression, make_classification,
make_blobs

print("=" * 70)
print("BAB 13 - MEN-GENERATE SYNTHETIC DATA DENGAN PYTHON")
print("=" * 70)

# ============================================================
# A. Data numerik buatan dengan NumPy
# ============================================================
print("\n[ A ] DATA NUMERIK DENGAN NUMPY")

np.random.seed(42)
jam_belajar = np.random.randint(1, 11, size=20)
noise = np.random.randint(-5, 6, size=20)
nilai = 50 + (jam_belajar * 4) + noise

df_numpy = pd.DataFrame({
    "jam_belajar": jam_belajar,
    "nilai": nilai
})

print(df_numpy.head())

plt.figure(figsize=(8, 5))
plt.scatter(df_numpy["jam_belajar"], df_numpy["nilai"])
plt.title("Synthetic Data dengan NumPy")
plt.xlabel("Jam Belajar")
plt.ylabel("Nilai")
plt.tight_layout()
plt.show()

# ============================================================
# B. make_regression
# ============================================================
print("\n[ B ] MAKE_REGRESSION")

X_reg, y_reg = make_regression(
    n_samples=100,
    n_features=1,
```

```
    n_informative=1,
    noise=10,
    random_state=42
)

df_reg = pd.DataFrame({
    "fitur_1": X_reg[:, 0],
    "target": y_reg
})

print(df_reg.head())

plt.figure(figsize=(8, 5))
plt.scatter(df_reg["fitur_1"], df_reg["target"])
plt.title("Synthetic Regression Data")
plt.xlabel("Fitur 1")
plt.ylabel("Target")
plt.tight_layout()
plt.show()

# ============================================================
# C. make_classification
# ============================================================
print("\n[ C ] MAKE_CLASSIFICATION")

X_cls, y_cls = make_classification(
    n_samples=200,
    n_features=2,
    n_informative=2,
    n_redundant=0,
    n_classes=2,
    random_state=42
)

df_cls = pd.DataFrame(X_cls, columns=["fitur_1", "fitur_2"])
df_cls["kelas"] = y_cls

print(df_cls.head())

plt.figure(figsize=(8, 5))
plt.scatter(df_cls["fitur_1"], df_cls["fitur_2"], c=df_cls["kelas"])
plt.title("Synthetic Classification Data")
plt.xlabel("Fitur 1")
plt.ylabel("Fitur 2")
plt.tight_layout()
plt.show()

# ============================================================
# D. make_blobs
# ============================================================
print("\n[ D ] MAKE_BLOBS")

X_blob, y_blob = make_blobs(
    n_samples=300,
    centers=3,
    n_features=2,
    cluster_std=1.2,
    random_state=42
)
```

```
df_blob = pd.DataFrame(X_blob, columns=["x1", "x2"])
df_blob["cluster"] = y_blob

print(df_blob.head())

plt.figure(figsize=(8, 5))
plt.scatter(df_blob["x1"], df_blob["x2"], c=df_blob["cluster"])
plt.title("Synthetic Clustering Data")
plt.xlabel("x1")
plt.ylabel("x2")
plt.tight_layout()
plt.show()

# ============================================================
# E. Menambah noise
# ============================================================
print("\n[ E ] MENAMBAH NOISE")

x = np.linspace(0, 10, 100)
y_ideal = 3 * x + 10
noise_realistis = np.random.normal(0, 5, size=100)
y_noisy = y_ideal + noise_realistis

plt.figure(figsize=(8, 5))
plt.scatter(x, y_noisy, label="Dengan noise")
plt.plot(x, y_ideal, label="Pola ideal")
plt.title("Simulasi Noise pada Data Regresi")
plt.xlabel("x")
plt.ylabel("y")
plt.legend()
plt.tight_layout()
plt.show()

print("\nSELESAI. Anda sudah membuat synthetic data untuk regresi,
klasifikasi, dan clustering.")
print("=" * 70)
```

13.9 Kelebihan *synthetic data*

Synthetic data punya banyak kelebihan, terutama untuk pembelajaran dan eksperimen awal.

Kelebihan utama

- **mudah dibuat**
- **cepat direproduksi**
- **aman dari sisi privasi**
- **cocok untuk latihan algoritma**
- **mudah dikontrol tingkat kesulitannya**
- **bisa dipakai untuk menguji pipeline atau kode**
- **baik untuk simulasi kondisi tertentu**

Dalam beberapa konteks yang lebih lanjut, data sintetis juga bisa membantu memperkaya data latih atau mengatasi keterbatasan contoh tertentu, seperti yang disinggung pada referensi GAN untuk keamanan siber.

Kelebihan untuk pemula

Untuk pembaca pemula, nilai terbesar *synthetic data* adalah:

- bisa fokus memahami konsep
- tidak langsung dibebani data kotor
- bisa melihat efek parameter secara jelas
- bisa melakukan eksperimen berulang dengan cepat

13.10 Keterbatasan *synthetic data*

Walaupun sangat berguna, *synthetic data* juga punya batas.

Keterbatasan utama

- **tidak selalu mewakili kompleksitas dunia nyata**
- bisa terlalu rapi dan terlalu ideal
- tidak menangkap seluruh bias, missing value, atau keanehan data nyata
- model yang terlihat bagus di data sintetis belum tentu bagus di data nyata
- hasil interpretasi bisa terlalu optimistis

Pelajaran penting

Synthetic data sangat bagus untuk:

- belajar
- uji coba
- eksplorasi awal
- demonstrasi algoritma

Tetapi tetap harus diingat bahwa **tujuan akhirnya adalah siap menghadapi data dunia nyata** yang lebih tidak rapi, lebih noisy, dan lebih ambigu.

13.11 Kapan synthetic data sangat membantu

Secara praktis, pembaca bisa memakai *synthetic data* saat:

- ingin belajar regresi tanpa harus mencari dataset dulu
- ingin memahami klasifikasi dua kelas
- ingin belajar clustering dan melihat bentuk cluster
- ingin menguji visualisasi
- ingin menguji preprocessing atau model baseline
- ingin membandingkan pengaruh noise kecil dan besar

Ini membuat proses belajar menjadi lebih **fun**, lebih **terarah**, dan lebih **menantang** secara sehat.

Penutup Bab

Bab ini menunjukkan bahwa **membuat data buatan adalah keterampilan belajar yang sangat kuat**. Dengan Python, NumPy, dan scikit-learn, pembaca bisa membuat *synthetic data* untuk regresi, klasifikasi, dan clustering, lalu menambahkan noise agar simulasi lebih realistis. Ini sangat cocok dengan semangat buk ini: belajar bertahap, *hands-on*, dan membangun intuisi sebelum masuk ke proyek yang lebih kompleks. Dukungan dari materi lampiran juga kuat, terutama pada penggunaan NumPy sebagai fondasi komputasi, scikit-learn sebagai alat praktis untuk tugas ML, dan pentingnya preprocessing serta noise handling dalam data yang lebih realistis.

Pesan terpenting dari bab ini: *synthetic data* bukan pengganti semua data nyata, tetapi ia adalah **laboratorium belajar** yang sangat berguna untuk membangun pemahaman model dengan cepat dan terkontrol.

Tantangan kecil untuk pembaca: ubah file `bab13_synthetic_data_terpadu.py` agar setiap generator membuat dua versi dataset: satu dengan noise rendah dan satu dengan noise tinggi, lalu bandingkan secara visual mana yang lebih mudah dipelajari model.

Bab 14. Feature Engineering Dasar

Bab ini adalah **tahap yang sangat penting** sebelum pembaca masuk lebih jauh ke model *machine learning*. Setelah memahami *synthetic data*, statistik, EDA, dan regresi, pembaca perlu belajar satu keterampilan inti yang sering membedakan model biasa dan model yang lebih baik, yaitu **feature engineering**. Di outline naskah utama, bab ini memang diletakkan sebelum inti *machine learning*, dengan fokus pada **memilih feature yang relevan, membuat feature turunan sederhana, encoding kategori, scaling, split train/test, dan menghindari data leakage**. Arah ini sangat selaras dengan materi lampiran yang menekankan bahwa **data preparation** dan **feature engineering** adalah fondasi yang langsung mempengaruhi kualitas model.

Dokumen referensi juga sangat mendukung susunan bab ini. Jain menjelaskan bahwa **feature engineering** adalah proses membuat atau memodifikasi fitur agar model bekerja lebih baik, termasuk **encoding kategorikal**, **feature scaling**, **interaction features**, dan **train-test split**. Van Der Post menekankan bahwa feature engineering adalah upaya kreatif untuk mengubah data mentah menjadi input yang lebih bermakna, termasuk membuat fitur interaksi dan agregasi yang membuka pola tersembunyi. Referensi lain juga menegaskan pentingnya memisahkan data latih dan data uji dengan benar serta melakukan *scaling* secara hati-hati agar evaluasi model tetap andal.

Bab ini sengaja dibuat **hands-on**, karena tambahan *synthetic data* memang sangat cocok untuk latihan model saat pembaca belum punya dataset riil yang rapi. Dengan data buatan atau data sederhana, pembaca bisa fokus memahami **bagaimana fitur dipilih, dibentuk, diubah, dan disiapkan** sebelum masuk ke model.

14.1 Mengapa *feature engineering* penting

Dalam *machine learning*, model tidak “melihat dunia” secara langsung. Model hanya melihat **fitur** yang kita berikan. Kalau fitur yang dipakai buruk, tidak relevan, salah skala, atau bocor dari masa depan, maka model bisa tampak cerdas tetapi sebenarnya rapuh. Karena itu, *feature engineering* bukan pelengkap, melainkan **jantung dari data preparation**. Referensi lampiran menegaskan bahwa kualitas data dan cara kita melakukan preprocessing secara langsung mempengaruhi performa model.

Secara praktis, *feature engineering* membantu kita:

- memilih kolom yang masuk akal
- membuang fitur yang tidak relevan
- membuat fitur baru yang lebih informatif
- mengubah data kategori menjadi numerik
- menyamakan skala fitur
- memisahkan data latih dan uji dengan sehat
- menghindari kebocoran informasi dari masa depan atau dari data uji

Contoh intuitif

Misalnya kita ingin memprediksi **nilai mahasiswa**. Beberapa fitur mentah mungkin:

- `jam_belajar`
- `kehadiran`
- `tugas`
- `kota`
- `nama`

Dari sini kita perlu berpikir:

- apakah `nama` relevan untuk prediksi? biasanya tidak
- apakah `kota` perlu di-*encode*? mungkin ya
- apakah kita bisa membuat fitur baru seperti `rasio_tugas_per_jam`? mungkin menarik
- apakah `jam_belajar` dan `kehadiran` perlu di-*scale*? tergantung modelnya

Di sinilah *feature engineering* bekerja.

14.2 Dataset contoh untuk bab ini

Agar bab ini benar-benar praktis, kita akan memakai dataset sederhana tentang mahasiswa. Dataset ini cukup kaya untuk latihan memilih fitur, membuat fitur turunan, *encoding*, *scaling*, *split train/test*, dan menjelaskan *data leakage*.

Simpan sebagai:

```
data/raw/feature_engineering_mahasiswa.csv
```

Isi file:

```
id,nama,kota,jam_belajar,kehadiran,tugas,nilai,status_kelulusan
1,Andi,Jakarta,6,82,78,80,Lulus
2,Budi,Bandung,7,85,80,84,Lulus
3,Citra,Depok,8,88,84,89,Lulus
4,Dina,Jakarta,5,79,74,76,Lulus
5,Eka,Bandung,9,90,88,92,Lulus
6,Fajar,Depok,10,94,91,95,Lulus
7,Gina,Jakarta,4,75,70,72,Tidak
8,Hadi,Surabaya,7,86,81,85,Lulus
9,Ika,Jakarta,8,89,86,90,Lulus
10,Joko,Bandung,5,78,73,75,Lulus
11,Kiki,Depok,6,83,79,81,Lulus
12,Lina,Jakarta,11,96,94,98,Lulus
13,Mira,Bandung,6,81,77,79,Lulus
14,Nanda,Depok,7,84,80,83,Lulus
15,Oki,Depok,3,72,68,69,Tidak
16,Prita,Jakarta,9,91,89,93,Lulus
17,Qori,Jakarta,6,82,78,80,Lulus
```

```
18,Raka,Bandung,10,93,90,96,Lulus
19,Sinta,Surabaya,8,87,85,88,Lulus
20,Tono,Jakarta,4,74,69,71,Tidak
```

Dataset ini memungkinkan kita melakukan dua jenis tugas:

- **regresi** → target `nilai`
- **klasifikasi** → target `status_kelulusan`

14.3 Memilih *feature* yang relevan

Apa itu *feature selection* secara dasar

Memilih *feature* yang relevan berarti menentukan kolom mana yang layak dipakai model. Tidak semua kolom harus masuk. Bahkan, kadang terlalu banyak fitur justru memperburuk model. Referensi Van Der Post menekankan pentingnya menyeimbangkan **kompleksitas** dan **interpretabilitas**, serta terus bereksperimen untuk melihat kontribusi fitur terhadap performa model.

Prinsip memilih fitur

Secara praktis, fitur yang baik biasanya:

- punya hubungan yang masuk akal dengan target
- tersedia saat prediksi dilakukan
- tidak terlalu banyak missing value
- tidak bersifat identitas murni seperti `id`
- tidak bocor dari target

Contoh memilih fitur dari dataset mahasiswa

Kalau targetnya `nilai`, maka kandidat fitur yang relevan:

- `jam_belajar`
- `kehadiran`
- `tugas`
- `kota`

Sedangkan fitur yang biasanya tidak relevan:

- `id`
- `nama`

Source code lengkap

Simpan sebagai:

```
src/feature_selection_dasar.py
```

Isi file:

```
import pandas as pd

df = pd.read_csv("data/raw/feature_engineering_mahasiswa.csv")

print("=== KOLOM DATASET ===")
print(df.columns.tolist())

# Pilih fitur yang relevan
fitur = ["jam_belajar", "kehadiran", "tugas", "kota"]
target_regresi = "nilai"
target_klasifikasi = "status_kelulusan"

X = df[fitur]
y_reg = df[target_regresi]
y_cls = df[target_klasifikasi]

print("\n=== FITUR TERPILIH ===")
print(X.head())

print("\n=== TARGET REGRESI ===")
print(y_reg.head())

print("\n=== TARGET KLASIFIKASI ===")
print(y_cls.head())
```

Latihan berpikir

Pembaca perlu membiasakan diri bertanya:

- apakah fitur ini tersedia saat prediksi nyata dilakukan?
- apakah fitur ini terlalu dekat dengan target?
- apakah fitur ini hanya penanda identitas?

14.4 Membuat *feature* turunan sederhana

Ini bagian yang sering sangat kuat. Referensi Jain menyebut **interaction features** dan **binning** sebagai contoh feature engineering, sedangkan Van Der Post memberi contoh fitur turunan seperti `AveragePurchaseInterval` dan interaksi `Age_Tenure_Interaction`. Artinya, membuat fitur baru dari fitur lama adalah praktik yang sangat umum dan sangat berguna.

Contoh fitur turunan pada dataset mahasiswa

Dari data yang ada, kita bisa membuat:

- `rata_aktivitas` = rata-rata dari `jam_belajar`, `kehadiran`, dan `tugas`
- `rasio_tugas_per_jam` = `tugas / jam_belajar`
- `interaksi_belajar_kehadiran` = `jam_belajar * kehadiran`
- `kategori_jam_belajar` = hasil *binning* jam belajar

Source code lengkap

Simpan sebagai:

```
src/feature_turunan_dasar.py
```

Isi file:

```
import pandas as pd

df = pd.read_csv("data/raw/feature_engineering_mahasiswa.csv")

# Feature turunan sederhana
df["rata_aktivitas"] = (df["jam_belajar"] + df["kehadiran"] +
df["tugas"]) / 3
df["rasio_tugas_per_jam"] = df["tugas"] / df["jam_belajar"]
df["interaksi_belajar_kehadiran"] = df["jam_belajar"] *
df["kehadiran"]

# Binning jam belajar
df["kategori_jam_belajar"] = pd.cut(
    df["jam_belajar"],
    bins=[0, 4, 7, 12],
    labels=["Rendah", "Sedang", "Tinggi"]
)

print(df[[
    "jam_belajar", "kehadiran", "tugas",
    "rata_aktivitas", "rasio_tugas_per_jam",
    "interaksi_belajar_kehadiran", "kategori_jam_belajar"
]].head(10))
```

Kapan fitur turunan berguna

Fitur turunan berguna ketika:

- ada hubungan yang tidak langsung tertangkap fitur mentah
- kita ingin memberi model sinyal yang lebih kuat
- kita ingin memasukkan pengetahuan domain sederhana
- kita ingin mengubah angka mentah menjadi bentuk yang lebih informatif

Catatan penting

Jangan membuat fitur turunan hanya karena “bisa”. Selalu tanya:

apakah fitur baru ini masuk akal secara konteks?

14.5 *Encoding* kategori

Sebagian model *machine learning* hanya menerima input numerik. Karena itu, fitur kategori seperti `kota` atau target seperti `status_kelulusan` perlu diubah menjadi angka. Referensi Jain menegaskan bahwa variabel kategorikal harus di-*encode*, biasanya melalui **one-hot encoding** atau **label encoding**.

A. One-hot encoding untuk fitur kategori

Untuk fitur seperti `kota`, pendekatan yang aman adalah **one-hot encoding**.

Simpan sebagai:

```
src/encoding_kategori_fitur.py
```

Isi file:

```
import pandas as pd

df = pd.read_csv("data/raw/feature_engineering_mahasiswa.csv")
df_encoded = pd.get_dummies(df, columns=["kota"], dtype=int)
print(df_encoded.head())
```

Hasilnya akan menambah kolom seperti:

- `kota_Bandung`
- `kota_Depok`
- `kota_Jakarta`
- `kota_Surabaya`

B. Label encoding sederhana untuk target biner

Untuk target klasifikasi `status_kelulusan`, kita bisa pakai pemetaan manual sederhana.

```
import pandas as pd

df = pd.read_csv("data/raw/feature_engineering_mahasiswa.csv")

mapping = {
    "Tidak": 0,
    "Lulus": 1
}

df["status_kelulusan_encoded"] = df["status_kelulusan"].map(mapping)

print(df[["status_kelulusan", "status_kelulusan_encoded"]].head())
```

Kapan pakai yang mana

- pakai **one-hot encoding** untuk fitur nominal seperti kota
- pakai **label encoding** atau pemetaan manual untuk target biner yang jelas seperti Lulus/Tidak

Catatan praktis

Referensi Uddin juga menyinggung *one-hot encoding* sebagai bagian dari feature engineering untuk membuat data lebih dapat diterima model.

14.6 *Scaling*

Mengapa scaling penting

Referensi Jain menegaskan bahwa *scaling* penting agar semua fitur berkontribusi lebih seimbang. Dua teknik utama adalah:

- **Standardization** dengan `StandardScaler`
- **Normalization** dengan `MinMaxScaler`

Model yang sensitif terhadap skala

Scaling sangat penting terutama untuk model seperti:

- KNN
- SVM
- K-Means
- regresi dengan regularisasi

A. StandardScaler

Simpan sebagai:

src/scaling_standard.py

Isi file:

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

df = pd.read_csv("data/raw/feature_engineering_mahasiswa.csv")

fitur_numerik = ["jam_belajar", "kehadiran", "tugas"]

scaler = StandardScaler()
scaled = scaler.fit_transform(df[fitur_numerik])

df_scaled = pd.DataFrame(scaled, columns=[f"{col}_std" for col
in fitur_numerik])
```

```
print(df_scaled.head())
```

B. MinMaxScaler

Simpan sebagai:

```
src/scaling_minmax.py
```

Isi file:

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

df = pd.read_csv("data/raw/feature_engineering_mahasiswa.csv")

fitur_numerik = ["jam_belajar", "kehadiran", "tugas"]

scaler = MinMaxScaler()
scaled = scaler.fit_transform(df[fitur_numerik])

df_scaled = pd.DataFrame(scaled, columns=[f"{col}_minmax" for
col in fitur_numerik])

print(df_scaled.head())
```

Kapan pakai standardisasi, kapan normalisasi

- **StandardScaler**: bagus saat ingin mean = 0 dan std = 1
- **MinMaxScaler**: bagus saat ingin rentang 0–1

Referensi Lim juga menyinggung penggunaan `MinMaxScaler` dan menekankan pentingnya melakukan scaling secara hati-hati terhadap data latih dan data uji.

14.7 *Split train/test*

Mengapa perlu *train-test split*

Kalau model dilatih dan diuji pada data yang sama, hasilnya cenderung terlalu optimistis. Karena itu, kita harus memisahkan data menjadi:

- **data train** → untuk melatih model
- **data test** → untuk mengevaluasi model

Referensi Jain menegaskan bahwa *train-test split* adalah langkah penting agar performa model diukur pada data yang tidak dilihat saat pelatihan. Ia juga menyebut rasio umum 80/20.

Source code lengkap

Simpan sebagai:

```
src/train_test_split_dasar.py
```

Isi file:

```
import pandas as pd
from sklearn.model_selection import train_test_split

df = pd.read_csv("data/raw/feature_engineering_mahasiswa.csv")

# Encoding sederhana
df["status_kelulusan_encoded"] =
df["status_kelulusan"].map({"Tidak": 0, "Lulus": 1})
df = pd.get_dummies(df, columns=["kota"], dtype=int)

fitur = ["jam_belajar", "kehadiran", "tugas", "kota_Bandung",
"kota_Depok", "kota_Jakarta", "kota_Surabaya"]
X = df[fitur]
y = df["status_kelulusan_encoded"]

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

print("Ukuran X_train:", X_train.shape)
print("Ukuran X_test :", X_test.shape)
print("Ukuran y_train:", y_train.shape)
print("Ukuran y_test :", y_test.shape)
```

Catatan penting

Untuk data deret waktu, *split* kadang tidak boleh acak. Referensi Lim menekankan bahwa pada *financial time series*, train dan test harus dipisah berdasarkan urutan waktu agar masa depan tidak bocor ke masa lalu.

14.8 Menghindari *data leakage*

Ini bagian yang sangat penting. Banyak model terlihat bagus bukan karena benar-benar pintar, tetapi karena mengalami **data leakage**.

Apa itu data leakage

Data leakage terjadi ketika informasi yang seharusnya tidak tersedia saat prediksi **bocor** ke proses pelatihan model. Akibatnya, model tampak sangat akurat, tetapi gagal saat dipakai di data nyata.

Contoh kebocoran yang sering terjadi

- melakukan scaling pada seluruh dataset sebelum split
- melakukan imputasi menggunakan seluruh data sebelum split
- memakai fitur yang sebenarnya berasal dari target
- memasukkan informasi masa depan
- melakukan feature engineering yang menggunakan statistik dari data uji

Referensi Lim memberi contoh yang sangat penting: pada data keuangan deret waktu, train dan test harus dipisah dengan benar, dan scaling perlu dilakukan dengan hati-hati agar informasi masa depan tidak mencemari pelatihan.

Contoh praktik salah

```
# CONTOH YANG SALAH
from sklearn.preprocessing import StandardScaler
from sklearn.model_selection import train_test_split

scaler = StandardScaler()
X_scaled = scaler.fit_transform(X)   # SALAH jika dilakukan sebelum
split
X_train, X_test, y_train, y_test = train_test_split(X_scaled, y,
test_size=0.2, random_state=42)
```

Contoh praktik yang benar

```
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler

df = pd.read_csv("data/raw/feature_engineering_mahasiswa.csv")
df = pd.get_dummies(df, columns=["kota"], dtype=int)

fitur = ["jam_belajar", "kehadiran", "tugas", "kota_Bandung",
"kota_Depok", "kota_Jakarta", "kota_Surabaya"]
X = df[fitur]
y = df["nilai"]

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

# Fit scaler hanya pada train
scaler = StandardScaler()
X_train_scaled = scaler.fit_transform(X_train)
X_test_scaled = scaler.transform(X_test)

print("X_train_scaled shape:", X_train_scaled.shape)
print("X_test_scaled shape :", X_test_scaled.shape)
```

Inti kebiasaan sehat

Urutan yang aman biasanya:

- split dulu
- fit preprocessing hanya pada train
- transform train
- transform test dengan objek yang sama

14.9 Pipeline preprocessing sederhana

Walaupun bab ini masih dasar, sangat baik jika pembaca mulai melihat bagaimana preprocessing bisa dirapikan dalam satu *pipeline*. Referensi Van Der Post memberi contoh `Pipeline` dan `ColumnTransformer` yang menggabungkan imputasi, scaling, dan one-hot encoding dalam satu alur konsisten.

Source code lengkap

Simpan sebagai:

```
src/pipeline_feature_engineering_dasar.py
```

Isi file:

```
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.compose import ColumnTransformer
from sklearn.pipeline import Pipeline
from sklearn.preprocessing import StandardScaler, OneHotEncoder
from sklearn.impute import SimpleImputer

df = pd.read_csv("data/raw/feature_engineering_mahasiswa.csv")

X = df[["jam_belajar", "kehadiran", "tugas", "kota"]]
y = df["nilai"]

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

numeric_features = ["jam_belajar", "kehadiran", "tugas"]
categorical_features = ["kota"]

numeric_transformer = Pipeline(steps=[
    ("imputer", SimpleImputer(strategy="median")),
    ("scaler", StandardScaler())
])

categorical_transformer = Pipeline(steps=[
```

```
        ("imputer", SimpleImputer(strategy="most_frequent")),
        ("onehot", OneHotEncoder(handle_unknown="ignore"))
    ])

    preprocessor = ColumnTransformer(
        transformers=[
            ("num", numeric_transformer, numeric_features),
            ("cat", categorical_transformer, categorical_features)
        ]
    )

    X_train_ready = preprocessor.fit_transform(X_train)
    X_test_ready = preprocessor.transform(X_test)

    print("Bentuk X_train_ready:", X_train_ready.shape)
    print("Bentuk X_test_ready :", X_test_ready.shape)
```

Mengapa ini bagus untuk pemula

Karena pembaca mulai melihat pola yang sehat:

- numerik diperlakukan berbeda dari kategorikal
- fit hanya dilakukan pada train
- test hanya di-transform
- alur kerja lebih rapi dan lebih aman dari *leakage*

14.10 Source code Python lengkap: latihan terpadu Bab 14

Simpan sebagai:

```
src/bab14_feature_engineering_terpadu.py
```

Isi file:

```
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler

print("=" * 80)
print("BAB 14 - FEATURE ENGINEERING DASAR")
print("=" * 80)

# ============================================================
# 1. LOAD DATA
# ============================================================
df = pd.read_csv("data/raw/feature_engineering_mahasiswa.csv")

print("\n[1] DATA AWAL")
print(df.head())

# ============================================================
# 2. PILIH FITUR RELEVAN
# ============================================================
fitur = ["jam_belajar", "kehadiran", "tugas", "kota"]
```

```
target_regresi = "nilai"
target_klasifikasi = "status_kelulusan"

print("\n[2] FITUR TERPILIH:", fitur)

# ============================================================
# 3. FEATURE TURUNAN
# ============================================================
df["rata_aktivitas"] = (df["jam_belajar"] + df["kehadiran"] + df["tugas"]) /
3
df["rasio_tugas_per_jam"] = df["tugas"] / df["jam_belajar"]
df["interaksi_belajar_kehadiran"] = df["jam_belajar"] * df["kehadiran"]

print("\n[3] FEATURE TURUNAN")
print(df[[
    "jam_belajar", "kehadiran", "tugas",
    "rata_aktivitas", "rasio_tugas_per_jam",
    "interaksi_belajar_kehadiran"
]].head())

# ============================================================
# 4. ENCODING TARGET DAN FITUR KATEGORI
# ============================================================
df["status_kelulusan_encoded"] = df["status_kelulusan"].map({"Tidak": 0,
"Lulus": 1})
df = pd.get_dummies(df, columns=["kota"], dtype=int)

print("\n[4] SETELAH ENCODING")
print(df.head())

# ============================================================
# 5. SIAPKAN FITUR AKHIR
# ============================================================
fitur_akhir = [
    "jam_belajar", "kehadiran", "tugas",
    "rata_aktivitas", "rasio_tugas_per_jam",
    "interaksi_belajar_kehadiran",
    "kota_Bandung", "kota_Depok", "kota_Jakarta", "kota_Surabaya"
]

X = df[fitur_akhir]
y_reg = df["nilai"]
y_cls = df["status_kelulusan_encoded"]

# ============================================================
# 6. TRAIN-TEST SPLIT
# ============================================================
X_train, X_test, y_train, y_test = train_test_split(
    X, y_reg,
    test_size=0.2,
    random_state=42
)

print("\n[5] TRAIN-TEST SPLIT")
print("X_train:", X_train.shape)
print("X_test :", X_test.shape)

# ============================================================
# 7. SCALING YANG BENAR (ANTI DATA LEAKAGE)
```

```
# ============================================================
scaler = StandardScaler()

X_train_scaled = scaler.fit_transform(X_train)
X_test_scaled = scaler.transform(X_test)

print("\n[6] SCALING SELESAI")
print("X_train_scaled shape:", X_train_scaled.shape)
print("X_test_scaled shape :", X_test_scaled.shape)

print("\nSELESAI. Anda sudah menjalankan feature engineering dasar dengan
alur yang sehat.")
print("=" * 80)
```

14.11 Urutan kerja yang disarankan

Agar pembaca membangun kebiasaan yang sehat, berikut urutan kerja yang saya sarankan.

- **Langkah 1:** pahami target prediksi dulu
- **Langkah 2:** pilih fitur yang relevan
- **Langkah 3:** buat fitur turunan jika masuk akal
- **Langkah 4:** lakukan encoding untuk kategori
- **Langkah 5:** split train/test lebih dulu
- **Langkah 6:** fit preprocessing hanya pada train
- **Langkah 7:** transform test menggunakan objek yang sama
- **Langkah 8:** dokumentasikan semua keputusan feature engineering

Urutan ini sangat sejalan dengan materi lampiran yang menempatkan feature engineering, split data, scaling, dan evaluasi sehat sebagai fondasi workflow *machine learning*.

14.12 Kesalahan umum pemula

Ada beberapa jebakan yang sangat sering terjadi.

- **memasukkan kolom identitas seperti `id` atau `nama`**
- **membuat fitur turunan tanpa logika konteks**
- **encoding target dan fitur tanpa membedakan kasus**
- **scaling sebelum split**
- **menggunakan statistik dari seluruh dataset untuk preprocessing**
- **membuat fitur yang secara tidak sadar memakai target**
- **tidak menyimpan alur preprocessing secara rapi**

Kebiasaan yang lebih baik adalah selalu bertanya:

> **apakah fitur ini benar-benar tersedia dan adil dipakai saat model nanti dipakai di dunia nyata?**

14.13 Refleksi: *feature engineering* adalah seni dan disiplin

Bab ini mengajarkan bahwa *feature engineering* bukan hanya pekerjaan teknis, tetapi gabungan antara:

- logika data
- intuisi domain
- kebersihan workflow
- kehati-hatian metodologis

Referensi Van Der Post secara eksplisit menyebut *feature engineering* sebagai usaha kreatif yang mengubah data mentah menjadi input bermakna, sementara Jain menekankan bahwa inilah salah satu langkah yang paling langsung memengaruhi kualitas model.

Dalam praktik nyata, model yang sederhana tetapi memakai fitur yang baik sering jauh lebih berguna daripada model rumit dengan fitur yang buruk. Karena itu, bab ini harus dipahami bukan sebagai bab tambahan, tetapi sebagai **pondasi kerja model yang sehat**.

Tambahan *synthetic data* yang sudah dibahas sebelumnya juga sangat cocok dengan orientasi *hands-on*, karena pembaca bisa berlatih *feature engineering* bahkan ketika belum punya dataset riil yang benar-benar rapi. Ini juga nyambung dengan banyak contoh lampiran yang memakai dataset sederhana atau contoh buatan untuk menjelaskan algoritma dan preprocessing.

Penutup Bab

Feature engineering dasar adalah **jembatan antara data mentah dan model yang layak dipercaya**. Melalui bab ini, pembaca telah belajar **memilih feature yang relevan**, **membuat feature turunan sederhana**, **encoding kategori**, **scaling**, **split train/test**, dan **menghindari data leakage**. Semua ini sangat selaras dengan outline naskah utama dan kuat didukung oleh materi lampiran yang menempatkan data preparation dan feature engineering sebagai fondasi langsung bagi performa model.

Pesan terpenting dari bab ini sederhana: **model belajar dari fitur yang Anda berikan, jadi kualitas berpikir Anda saat menyiapkan fitur sering lebih menentukan daripada pilihan algoritma itu sendiri**.

Tantangan kecil untuk pembaca: modifikasi file `bab14_feature_engineering_terpadu.py` agar selain fitur yang sudah ada, Anda membuat satu fitur baru bernama `disiplin_index`, misalnya gabungan antara `kehadiran` dan `tugas`, lalu siapkan dua versi dataset: satu dengan fitur lama saja, satu lagi dengan fitur baru, untuk dibandingkan pada bab modeling berikutnya.

Bagian IV — Machine Learning Inti dengan Banyak Contoh Kode

Bab 15. Memahami Machine Learning tanpa Takut Istilah

Bab ini berfungsi sebagai **gerbang mental** sebelum pembaca masuk ke algoritma satu per satu. Di outline buku utama, Bab 15 memang dirancang untuk menjelaskan **apa itu AI, ML, dan *deep learning***, lalu membedakan **supervised vs unsupervised**, **klasifikasi vs regresi**, **overfitting vs underfitting**, **train–validation–test**, dan **mengapa model harus dievaluasi, bukan sekadar dijalankan**. Struktur seperti ini juga sangat selaras dengan referensi lampiran yang menempatkan pengantar ML sebagai jembatan dari *data analysis* menuju *predictive modeling*.

Secara praktis, pembaca sering takut pada istilah *machine learning* karena terdengar rumit. Padahal inti awalnya cukup sederhana: **kita punya data, kita ingin menemukan pola, lalu memakai pola itu untuk memprediksi atau mengelompokkan data baru**. Referensi Lim menjelaskan *machine learning* sebagai pendekatan yang memetakan sekumpulan fitur `X` ke hasil `Y`, baik berupa angka maupun kelas, melalui proses optimisasi parameter model. Referensi Van Der Post juga menekankan bahwa *machine learning* belajar dari data historis untuk menghasilkan prediksi atau keputusan yang lebih baik pada data yang belum pernah dilihat sebelumnya.

Bab ini disusun agar pembaca mampu:

- **membedakan AI, ML, dan *deep learning***
- **memahami jenis masalah ML yang paling umum**
- **mengenali risiko model seperti overfitting dan underfitting**
- **memahami peran data train, validation, dan test**
- **mengerti mengapa evaluasi model itu wajib**
- **mencoba contoh kode Python sederhana** untuk melihat konsep-konsep ini secara nyata

15.1 AI, ML, dan *deep learning*

Banyak pemula mencampuradukkan ketiga istilah ini. Cara paling mudah memahaminya adalah sebagai **lingkaran bertingkat**.

- **AI (*Artificial Intelligence*)** adalah payung besar: segala pendekatan yang membuat komputer tampak “cerdas”.
- **ML (*Machine Learning*)** adalah bagian dari AI yang belajar dari data.
- **Deep learning** adalah bagian dari ML yang memakai jaringan saraf berlapis untuk menangani masalah yang lebih kompleks.

Referensi Saltz menjelaskan bahwa *deep learning* adalah penggunaan jaringan saraf berlapis (*layered neural networks*) untuk menyelesaikan masalah komputasi yang kompleks, dan bahwa teknik ini berdampak besar pada pengenalan suara, pengenalan gambar,

machine translation, dan sistem percakapan modern. Referensi Lim juga menempatkan *machine learning* sebagai domain AI yang berorientasi pada prediksi atau klasifikasi dari data, sedangkan *deep learning* menjadi cabang yang kuat terutama ketika data besar dan pola kompleks terlibat.

Ilustrasi intuitif

Bayangkan seperti ini:

- **AI** = tujuan besar membuat sistem cerdas
- **ML** = cara agar sistem belajar dari contoh
- **Deep learning** = cara ML yang memakai banyak lapisan neuron buatan

Contoh sehari-hari

- Filter spam email → AI/ML
- Prediksi harga rumah → ML
- Deteksi objek pada foto → sering *deep learning*
- Pengenalan ucapan seperti asisten suara → banyak memakai *deep learning*

Source code Python kecil: AI/ML bukan sihir

Kode berikut bukan *deep learning*, tetapi cukup untuk menunjukkan bahwa ML itu sebenarnya pola **fitur → target**.

Simpan sebagai:

```
src/bab15_ai_ml_pengantar.py

import pandas as pd
from sklearn.linear_model import LinearRegression

# Data sederhana: jam belajar -> nilai
data = {
    "jam_belajar": [2, 4, 6, 8, 10],
    "nilai": [60, 68, 76, 85, 93]
}

df = pd.DataFrame(data)

X = df[["jam_belajar"]]
y = df["nilai"]

model = LinearRegression()
model.fit(X, y)

prediksi = model.predict([[7]])

print("Koefisien :", model.coef_[0])
print("Intercept :", model.intercept_)
print("Prediksi nilai untuk 7 jam belajar:", prediksi[0])
```

Kode ini membantu pembaca melihat bahwa *machine learning* pada level awal hanyalah proses **belajar hubungan dari data**, bukan sesuatu yang mistis.

15.2 *Supervised* vs *unsupervised learning*

Ini adalah pembagian paling penting dalam pengantar ML. Referensi Jain secara eksplisit menempatkan **supervised vs unsupervised learning** sebagai inti Bab pengantar ML, dan Van Der Post menjelaskan keduanya dengan analogi yang sangat mudah dipahami: *supervised learning* seperti murid belajar dengan guru, sedangkan *unsupervised learning* seperti sistem menjelajahi data tanpa label jawaban.

A. *Supervised learning*

Pada *supervised learning*, kita punya:

- **fitur / input**
- **label / target / output**

Model belajar dari pasangan input-output itu.

Contoh:

- `jam_belajar -> nilai`
- `fitur rumah -> harga`
- `data nasabah -> default atau tidak`

B. *Unsupervised learning*

Pada *unsupervised learning*, kita hanya punya data input, **tanpa label target**. Model mencoba menemukan struktur tersembunyi.

Contoh:

- segmentasi pelanggan
- pengelompokan dokumen
- menemukan pola cluster pada data

Ringkasnya

- **supervised**: ada jawaban yang diajarkan
- **unsupervised**: tidak ada jawaban eksplisit, sistem mencari pola sendiri

Source code Python lengkap: contoh supervised sederhana

```
import pandas as pd
from sklearn.linear_model import LinearRegression

# Supervised learning: ada target
df = pd.DataFrame({
    "jam_belajar": [2, 4, 6, 8, 10],
    "nilai": [60, 68, 76, 85, 93]
})

X = df[["jam_belajar"]]
y = df["nilai"]

model = LinearRegression()
model.fit(X, y)

print("Prediksi nilai untuk 9 jam belajar:", model.predict([[9]])[0])
```

Source code Python lengkap: contoh unsupervised sederhana

```
import pandas as pd
from sklearn.cluster import KMeans

# Unsupervised learning: tidak ada target
df = pd.DataFrame({
    "belanja_bulanan": [1, 2, 2, 8, 9, 10],
    "frekuensi_kunjungan": [1, 1, 2, 8, 9, 10]
})

X = df[["belanja_bulanan", "frekuensi_kunjungan"]]

model = KMeans(n_clusters=2, random_state=42, n_init=10)
df["cluster"] = model.fit_predict(X)

print(df)
```

Kode kedua menunjukkan bahwa model bisa membentuk kelompok walau kita tidak memberi label “tipe pelanggan”.

15.3 Klasifikasi vs regresi

Ini adalah pembagian masalah **dalam supervised learning**. Referensi Jain menyebut **regression vs classification** sebagai pembedaan inti, dan Saltz menjelaskan bahwa regresi memprediksi output metrik/numerik, sedangkan klasifikasi memprediksi outcome kategorikal.

A. Regresi

Regresi dipakai jika target berbentuk **angka kontinu**.

Contoh:

- prediksi harga rumah
- prediksi penjualan
- prediksi konsumsi listrik
- prediksi nilai mahasiswa

B. Klasifikasi

Klasifikasi dipakai jika target berbentuk **kelas / label**.

Contoh:

- lulus / tidak lulus
- spam / tidak spam
- fraud / tidak fraud
- sehat / sakit

Ringkasan sederhana

- **regresi** → jawaban berupa angka
- **klasifikasi** → jawaban berupa kategori

Source code Python lengkap: regresi sederhana

```
import pandas as pd
from sklearn.linear_model import LinearRegression

df = pd.DataFrame({
    "jam_belajar": [2, 4, 6, 8, 10],
    "nilai": [60, 68, 76, 85, 93]
})

X = df[["jam_belajar"]]
y = df["nilai"]

model = LinearRegression()
model.fit(X, y)

print("Prediksi nilai untuk 7 jam belajar:", model.predict([[7]])[0])
```

Source code Python lengkap: klasifikasi sederhana

```
import pandas as pd
from sklearn.linear_model import LogisticRegression

df = pd.DataFrame({
    "jam_belajar": [2, 3, 4, 7, 8, 9],
    "lulus": [0, 0, 0, 1, 1, 1]
})

X = df[["jam_belajar"]]
y = df["lulus"]

model = LogisticRegression()
model.fit(X, y)

prediksi_kelas = model.predict([[6]])[0]
prediksi_prob = model.predict_proba([[6]])[0]

print("Prediksi kelas untuk 6 jam belajar:", prediksi_kelas)
print("Probabilitas [tidak lulus, lulus]:", prediksi_prob)
```

Kode ini menunjukkan bahwa pada klasifikasi, model sering memberi **kelas** dan juga **probabilitas kelas**.

15.4 *Overfitting* vs *underfitting*

Ini adalah dua istilah yang paling sering membuat pemula bingung, padahal konsepnya sangat penting. Referensi Van Der Post menjelaskan **overfitting** sebagai keadaan ketika model belajar terlalu detail sampai ikut “menghafal noise”, sedangkan **underfitting** terjadi ketika model terlalu sederhana sehingga gagal menangkap struktur data. Lim juga menjelaskan bahwa validasi dipakai untuk membantu mendeteksi masalah *over-fitting* pada training yang berujung performa buruk di validation/test.

A. Overfitting

Overfitting terjadi ketika model:

- sangat bagus di data train
- tetapi buruk di data baru

Model seperti ini mirip murid yang **menghafal jawaban**, bukan memahami konsep.

Tanda umum:

- akurasi train sangat tinggi
- akurasi validation/test jauh lebih rendah
- model terlalu kompleks

B. Underfitting

Underfitting terjadi ketika model:

- buruk di data train
- buruk juga di data validation/test

Model seperti ini terlalu sederhana, seperti murid yang bahkan belum mengerti materi dasar.

Ilustrasi intuitif

- **overfit** = terlalu hafal detail
- **underfit** = terlalu dangkal
- yang kita cari = **cukup pas untuk *generalize***

Source code Python lengkap: ilustrasi sederhana

```
import numpy as np
import matplotlib.pyplot as plt
from sklearn.preprocessing import PolynomialFeatures
from sklearn.linear_model import LinearRegression
from sklearn.pipeline import make_pipeline

# Data sederhana
np.random.seed(42)
X = np.linspace(0, 10, 20).reshape(-1, 1)
y = 3 * X.squeeze() + 10 + np.random.normal(0, 2, size=20)

# Model terlalu sederhana (underfit untuk data non-linear; di sini
contoh konsep)
model_linear = LinearRegression()
model_linear.fit(X, y)

# Model terlalu kompleks (potensi overfit)
model_poly = make_pipeline(PolynomialFeatures(degree=10),
LinearRegression())
model_poly.fit(X, y)

X_plot = np.linspace(0, 10, 200).reshape(-1, 1)
y_linear = model_linear.predict(X_plot)
y_poly = model_poly.predict(X_plot)

plt.figure(figsize=(9, 5))
plt.scatter(X, y, label="Data")
plt.plot(X_plot, y_linear, label="Model linear sederhana")
plt.plot(X_plot, y_poly, label="Model polynomial degree 10")
plt.title("Ilustrasi Underfitting vs Potensi Overfitting")
plt.xlabel("X")
plt.ylabel("y")
plt.legend()
plt.tight_layout()
```

```
plt.show()
```

Pelajaran dari contoh ini

- model terlalu sederhana bisa gagal menangkap pola
- model terlalu rumit bisa mengejar noise
- tugas kita adalah mencari model yang **cukup baik untuk data baru**, bukan hanya bagus di data latih

15.5 Train, validation, test

Ini adalah fondasi evaluasi model yang sehat. Referensi Lim menjelaskan dengan cukup jelas bahwa alur umum ML membagi data menjadi **training**, **validation**, dan **test**. Training dipakai untuk melatih parameter model, validation untuk memilih model atau *hyperparameter* dan memantau *overfitting*, lalu test dipakai untuk evaluasi akhir pada data yang benar-benar tidak dipakai selama pengembangan.

A. Training set

Dipakai untuk **melatih model**.

Model belajar pola dari data ini.

B. Validation set

Dipakai untuk:

- membandingkan model
- memilih *hyperparameter*
- mendeteksi *overfitting*

Validation set penting jika kita sedang *tuning* model. Jika model sangat sederhana tanpa banyak pengaturan, kadang cukup train-test saja. Itu juga disinggung oleh Lim.

C. Test set

Dipakai untuk **evaluasi final**. Data ini seharusnya tidak disentuh selama proses pemilihan model.

Rasio umum

Sebagai panduan awal:

- train 60–80%
- validation 10–20%
- test 10–20%

Rasio persisnya tergantung ukuran dataset, tetapi prinsip utamanya tetap sama: **jangan menguji model pada data yang dipakai melatihnya**.

Source code Python lengkap: train-test split

```
import pandas as pd
from sklearn.model_selection import train_test_split

df = pd.DataFrame({
    "jam_belajar": [2, 3, 4, 5, 6, 7, 8, 9, 10, 11],
    "nilai": [60, 64, 68, 72, 76, 80, 84, 88, 92, 95]
})

X = df[["jam_belajar"]]
y = df["nilai"]

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

print("Ukuran X_train:", X_train.shape)
print("Ukuran X_test :", X_test.shape)
print("Ukuran y_train:", y_train.shape)
print("Ukuran y_test :", y_test.shape)
```

Source code Python lengkap: train-validation-test sederhana

```
import pandas as pd
from sklearn.model_selection import train_test_split

df = pd.DataFrame({
    "jam_belajar": [2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13],
    "nilai": [60, 64, 68, 72, 76, 80, 84, 88, 92, 95, 98, 100]
})

X = df[["jam_belajar"]]
y = df["nilai"]

# Pisah train dan sisanya
X_train, X_temp, y_train, y_temp = train_test_split(
    X, y,
    test_size=0.4,
    random_state=42
)

# Pisah validation dan test
X_val, X_test, y_val, y_test = train_test_split(
    X_temp, y_temp,
    test_size=0.5,
    random_state=42
)

print("Train:", X_train.shape, y_train.shape)
print("Validation:", X_val.shape, y_val.shape)
print("Test:", X_test.shape, y_test.shape)
```

15.6 Kenapa model harus dievaluasi

Ini mungkin pesan terpenting dalam bab ini. Di outline utama juga tertulis jelas bahwa model harus **dievaluasi, bukan hanya dijalankan**. Referensi Van Der Post menekankan bahwa evaluasi model adalah proses yang melampaui sekadar statistik mentah; ia menentukan apakah model sungguh mampu menjawab masalah nyata. Lim juga menegaskan bahwa prediksi pada test set harus dibandingkan dengan target aktual menggunakan metrik performa, dan jika performanya tidak baik, model perlu dipertimbangkan ulang.

Mengapa evaluasi wajib

Karena model yang “jalan” belum tentu model yang “berguna”.

Tanpa evaluasi, kita tidak tahu:

- apakah model benar-benar belajar
- apakah model hanya menghafal
- apakah error-nya kecil atau besar
- apakah model layak dipakai

Metrik tergantung tugas

Untuk **regresi**, metrik umum:

- MAE
- MSE
- RMSE
- R^2

Untuk **klasifikasi**, metrik umum:

- accuracy
- precision
- recall
- F1-score
- confusion matrix

Referensi Jain menempatkan evaluasi performa seperti **MSE** dan **R^2** dalam bab model pertama, sementara referensi lain menekankan metrik klasifikasi seperti accuracy dan F1 untuk tugas klasifikasi.

Source code Python lengkap: evaluasi regresi

```
import pandas as pd
import math
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LinearRegression
from sklearn.metrics import mean_absolute_error, mean_squared_error,
r2_score

df = pd.DataFrame({
    "jam_belajar": [2, 3, 4, 5, 6, 7, 8, 9, 10, 11],
    "nilai": [60, 64, 68, 72, 76, 80, 84, 88, 92, 95]
})

X = df[["jam_belajar"]]
y = df["nilai"]

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

model = LinearRegression()
model.fit(X_train, y_train)

y_pred = model.predict(X_test)

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

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

Source code Python lengkap: evaluasi klasifikasi

```
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import accuracy_score, classification_report,
confusion_matrix

df = pd.DataFrame({
    "jam_belajar": [2, 3, 4, 5, 6, 7, 8, 9, 10, 11],
    "lulus":       [0, 0, 0, 0, 1, 1, 1, 1, 1, 1]
})

X = df[["jam_belajar"]]
y = df["lulus"]

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

model = LogisticRegression()
model.fit(X_train, y_train)

y_pred = model.predict(X_test)

print("Accuracy:", accuracy_score(y_test, y_pred))
print("\nConfusion Matrix:")
print(confusion_matrix(y_test, y_pred))
print("\nClassification Report:")
print(classification_report(y_test, y_pred))
```

15.7 Contoh alur ML sederhana dari awal sampai evaluasi

Bagian ini penting agar pembaca melihat gambaran besar.

Simpan sebagai:

```
src/bab15_alur_ml_sederhana.py

import pandas as pd
import math
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LinearRegression
from sklearn.metrics import mean_absolute_error, mean_squared_error,
r2_score

print("=" * 70)
print("BAB 15 - MEMAHAMI MACHINE LEARNING TANPA TAKUT ISTILAH")
print("=" * 70)

# 1. Siapkan data
df = pd.DataFrame({
    "jam_belajar": [2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12],
    "nilai": [60, 64, 68, 72, 76, 80, 84, 88, 92, 95, 98]
})
```

```
print("\n[1] DATA AWAL")
print(df)

# 2. Pisahkan fitur dan target
X = df[["jam_belajar"]]
y = df["nilai"]

# 3. Split train-test
X_train, X_test, y_train, y_test = train_test_split(
    X, y,
    test_size=0.2,
    random_state=42
)

print("\n[2] UKURAN DATA")
print("X_train:", X_train.shape)
print("X_test :", X_test.shape)

# 4. Buat dan latih model
model = LinearRegression()
model.fit(X_train, y_train)

print("\n[3] PARAMETER MODEL")
print("Koefisien:", model.coef_[0])
print("Intercept:", model.intercept_)

# 5. Prediksi
y_pred = model.predict(X_test)

hasil = pd.DataFrame({
    "aktual": y_test.values,
    "prediksi": y_pred
})

print("\n[4] HASIL PREDIKSI")
print(hasil)

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

print("\n[5] EVALUASI")
print("MAE  :", mae)
print("MSE  :", mse)
print("RMSE :", rmse)
print("R²   :", r2)

print("\nSELESAI. Inti machine learning: data -> model -> prediksi ->
evaluasi.")
print("=" * 70)
```

Program ini memperlihatkan inti paling sederhana dari ML modern:
data → split → fit → predict → evaluate. Itu selaras dengan workflow yang juga ditekankan dalam referensi lampiran.

15.8 Cara berpikir yang sehat untuk pemula

Agar pembaca tidak takut istilah, saya sarankan memegang pola pikir berikut:

- **AI** adalah payung besarnya
- **ML** adalah cara belajar dari data
- ***deep learning*** adalah cabang ML yang lebih kuat tetapi lebih kompleks
- **supervised** berarti ada label
- **unsupervised** berarti tidak ada label
- **regresi** memprediksi angka
- **klasifikasi** memprediksi kelas
- **overfitting** berarti terlalu hafal
- **underfitting** berarti terlalu dangkal
- **train/validation/test** adalah cara menjaga evaluasi tetap jujur
- **evaluasi** adalah pembeda antara eksperimen yang menarik dan model yang benar-benar berguna

15.9 Kesalahan umum pemula

Ada beberapa jebakan yang sangat sering muncul.

- **menganggap AI, ML, dan deep learning sama persis**
- **tidak membedakan regresi dan klasifikasi**
- **menguji model pada data latih yang sama**
- **mengira model dengan skor train tinggi pasti bagus**
- **tidak paham peran validation**
- **mengabaikan evaluasi, lalu merasa model sudah selesai**
- **menganggap korelasi atau akurasi awal sebagai bukti final**

Kebiasaan yang lebih sehat adalah selalu bertanya:

> **apakah model ini benar-benar belajar pola yang berguna untuk data baru?**

15.10 Refleksi: ML bukan soal istilah, tetapi soal alur pikir

Bab ini seharusnya membantu pembaca merasa lebih tenang. Banyak istilah di *machine learning* terdengar menakutkan hanya karena disebut sekaligus. Padahal jika dipecah, semuanya membentuk satu alur yang masuk akal:

- ada masalah
- ada data
- ada target atau tidak ada target
- kita pilih jenis tugas
- kita latih model
- kita cek apakah model terlalu hafal atau terlalu dangkal
- kita evaluasi pada data yang belum pernah dilihat

Itulah inti dari ML. Referensi lampiran juga menunjukkan bahwa pengantar ML yang baik memang harus dimulai dari pembedaan-pembedaan dasar seperti **supervised/unsupervised**, **regression/classification**, **train/test**, dan **evaluasi model** sebelum masuk ke algoritma yang lebih spesifik.

Penutup Bab

Bab ini menegaskan bahwa *machine learning* tidak harus ditakuti. Dengan memahami **AI, ML, dan deep learning**, lalu membedakan **supervised vs unsupervised**, **klasifikasi vs regresi**, **overfitting vs underfitting**, **train/validation/test**, dan **mengapa model harus dievaluasi**, pembaca sudah memiliki fondasi mental yang sangat kuat sebelum masuk ke algoritma berikutnya. Ini sangat selaras dengan outline utama buku dan kuat didukung oleh materi lampiran yang menempatkan pengantar ML sebagai tahap penting sebelum implementasi.

Pesan terpenting dari bab ini: *machine learning* bukan sekadar menjalankan model, tetapi memahami **jenis masalah, jenis data, dan cara menilai apakah model benar-benar berguna**.

Tantangan kecil untuk pembaca: ubah file `bab15_alur_ml_sederhana.py` menjadi dua versi: satu versi **regresi** untuk memprediksi `nilai`, dan satu versi **klasifikasi** untuk memprediksi `lulus/tidak`, lalu bandingkan bagaimana *output* dan metrik evaluasinya berubah.

Bab 16. Workflow Machine Learning End-to-End

Bab ini adalah **peta kerja utama** untuk semua proyek *machine learning* yang sehat. Setelah pembaca memahami istilah-istilah dasar ML pada bab sebelumnya, kini fokusnya bergeser dari “apa itu ML” menjadi **bagaimana ML dikerjakan dari awal sampai akhir**. Di outline naskah utama, Bab 16 memang dirancang sebagai alur yang benar: **definisikan masalah, pahami target variable, siapkan feature, split data, buat baseline model, evaluasi, iterasi, dokumentasi eksperimen, dan pengantar pipeline di scikit-learn**. Ini menunjukkan bahwa proyek ML yang baik bukan sekadar memilih algoritma, tetapi menjalankan proses yang runtut dan disiplin.

Materi lampiran juga sangat kuat mendukung alur ini. Referensi *AI Agents with Python* menekankan bahwa setelah data dikumpulkan, tahap berikutnya adalah **preprocessing, feature engineering, dan split train-test** agar model bisa *generalize* pada data baru. Referensi *Python for Data Analysis* menyebut **data preparation, feature engineering, train-test splitting, cross-validation, dan scaling** sebagai fondasi workflow ML sebelum model pertama dibangun. Sementara referensi lain menegaskan pentingnya **testing, evaluasi metrik, iterasi, dan dokumentasi eksperimen** agar model tidak berhenti di satu kali percobaan.

Bab ini dirancang agar pembaca mampu menjalankan alur berikut secara utuh:

- **definisikan masalah**
- **tentukan target**
- **pilih feature**
- **split data**
- **bangun baseline**
- **latih model**
- **evaluasi**
- **iterasi**
- **simpan model sederhana**

Semua contoh akan dibuat **hands-on**, memakai Python dan scikit-learn, dengan kode yang bisa dijalankan di Ubuntu 26.04.

16.1 Mengapa workflow lebih penting daripada sekadar algoritma

Banyak pemula bertanya, “algoritma apa yang terbaik?” Padahal dalam praktik nyata, pertanyaan yang lebih sehat adalah:

> **apakah workflow saya sudah benar?**

Model yang sederhana tetapi dibangun dengan alur yang benar sering lebih berguna daripada model canggih yang dibangun dengan data bocor, evaluasi yang salah, atau target yang tidak jelas. Referensi lampiran juga menegaskan bahwa kualitas data, preprocessing, feature engineering, pembagian train-test, dan evaluasi yang benar sangat memengaruhi hasil model.

Dengan kata lain:

- **algoritma penting**
- tetapi **workflow lebih menentukan apakah hasilnya bisa dipercaya**

16.2 Gambaran besar alur *machine learning end-to-end*

Secara umum, workflow ML yang sehat bisa diringkas seperti ini:

1. **Definisikan masalah**
2. **Tentukan target**
3. **Pilih feature**
4. **Split data**
5. **Bangun baseline**
6. **Latih model**
7. **Evaluasi**
8. **Iterasi**
9. **Simpan model**

Ini sangat selaras dengan outline buku utama dan materi lampiran yang menyebutkan tahapan serupa berulang kali dalam proyek regresi, klasifikasi, maupun proyek AI sederhana.

Agar pembahasan terasa nyata, kita akan memakai dataset sederhana bertema prediksi nilai mahasiswa.

16.3 Dataset contoh

Simpan file berikut sebagai:

```
data/raw/workflow_ml_mahasiswa.csv
```

Isi file:

```
jam_belajar,kehadiran,tugas,nilai
2,70,60,58
3,75,65,63
4,78,68,67
5,80,70,71
6,82,75,75
7,85,78,79
8,88,82,84
9,90,85,88
10,92,88,91
11,95,90,94
12,96,94,97
5,79,72,70
6,81,74,74
7,84,77,78
8,87,81,83
```

```
9,89,84,87
10,91,87,90
11,94,89,93
```

Dataset ini cukup baik untuk menunjukkan workflow penuh:

- target: `nilai`
- feature: `jam_belajar`, `kehadiran`, `tugas`

16.4 Definisikan masalah

Langkah pertama dalam ML bukan coding, tetapi **mendefinisikan masalah dengan jelas**. Referensi Knox bahkan menekankan bahwa praktisi ML harus bisa memahami kebutuhan klien atau konteks masalah, karena tanpa definisi masalah yang benar, solusi teknis mudah meleset.

Pertanyaan yang harus dijawab

Sebelum membuka scikit-learn, tanyakan:

- apa yang ingin diprediksi?
- apakah targetnya angka atau kelas?
- siapa pengguna hasil model ini?
- apa keputusan yang akan dibantu model?
- apakah model dipakai untuk prediksi, segmentasi, atau ranking?

Contoh pada dataset mahasiswa

Masalah kita:

> **memprediksi nilai mahasiswa berdasarkan jam belajar, kehadiran, dan tugas**

Ini berarti:

- jenis masalah: **supervised learning**
- tipe target: **angka**
- tugas ML: **regresi**

Kalau definisi masalah salah, seluruh workflow bisa salah arah.

16.5 Tentukan target

Setelah masalah jelas, kita harus menentukan **target variable**. Outline utama secara eksplisit menulis “pahami target variable”, dan referensi Lim juga menekankan bahwa pada *supervised learning*, harus ada target/label yang jelas untuk diprediksi.

Pada contoh ini

- target = `nilai`

Itu berarti model akan mencoba memetakan fitur menjadi nilai prediksi.

Kesalahan umum

Pemula sering:

- salah memilih kolom target
- memakai target yang sebenarnya belum tersedia saat prediksi
- memakai kolom yang terlalu dekat dengan target sehingga berpotensi *leakage*

Karena itu, target harus:

- jelas
- relevan
- tersedia dalam data latih
- benar-benar mewakili pertanyaan bisnis/analisis

16.6 Pilih feature

Setelah target ditentukan, langkah berikutnya adalah memilih fitur yang masuk akal. Referensi *AI Agents with Python* dan *Python for Data Analysis* menekankan bahwa feature engineering dan pemilihan fitur sangat penting agar model punya input yang bermakna.

Pada contoh ini

Fitur yang dipilih:

- `jam_belajar`
- `kehadiran`
- `tugas`

Kenapa?

- ketiganya masuk akal secara konteks
- tersedia sebelum target diketahui
- relevan untuk prediksi nilai

Prinsip memilih fitur

Pilih fitur yang:

- masuk akal terhadap target
- tersedia saat prediksi dilakukan
- tidak terlalu bocor dari target
- tidak hanya berupa identitas

16.7 Split data

Langkah berikutnya adalah memisahkan data menjadi **train** dan **test**. Ini ditekankan kuat di banyak referensi lampiran. *AI Agents with Python* menyebut split 80/20 sebagai praktik umum agar model diuji pada data yang belum pernah dilihat. Jain juga menempatkan *train-test split* sebagai langkah fondasional sebelum model dibangun.

Kenapa split penting

Karena kita ingin tahu:

apakah model benar-benar bisa bekerja pada data baru?

Kalau data train dan test sama, hasil akan terlalu optimistis.

Source code Python lengkap

Simpan sebagai:

```
src/bab16_langkah_split.py

import pandas as pd
from sklearn.model_selection import train_test_split

df = pd.read_csv("data/raw/workflow_ml_mahasiswa.csv")

X = df[["jam_belajar", "kehadiran", "tugas"]]
y = df["nilai"]

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

print("X_train:", X_train.shape)
print("X_test :", X_test.shape)
print("y_train:", y_train.shape)
print("y_test :", y_test.shape)
```

16.8 Bangun baseline

Ini langkah yang sering dilupakan, padahal sangat penting. Outline utama secara eksplisit menulis **buat baseline model**. Tujuan baseline adalah memberi patokan awal: model sederhana apa yang bisa kita capai dulu sebelum mengejar model yang lebih rumit.

Kenapa baseline penting

Karena tanpa baseline, kita tidak tahu apakah model yang lebih rumit benar-benar lebih baik.

Bentuk baseline yang umum

Untuk regresi, baseline bisa berupa:

- prediksi selalu memakai **mean target**
- model linear sederhana
- model paling mudah diinterpretasikan

Source code Python lengkap: baseline mean

Simpan sebagai:

```
src/bab16_baseline_mean.py

import pandas as pd
import math
from sklearn.model_selection import train_test_split
from sklearn.metrics import mean_absolute_error, mean_squared_error,
r2_score

df = pd.read_csv("data/raw/workflow_ml_mahasiswa.csv")

X = df[["jam_belajar", "kehadiran", "tugas"]]
y = df["nilai"]

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

# Baseline: selalu prediksi mean dari y_train
baseline_pred = [y_train.mean()] * len(y_test)

mae = mean_absolute_error(y_test, baseline_pred)
mse = mean_squared_error(y_test, baseline_pred)
rmse = math.sqrt(mse)
r2 = r2_score(y_test, baseline_pred)

print("Baseline prediction (mean train):", y_train.mean())
print("MAE  :", mae)
print("MSE  :", mse)
print("RMSE :", rmse)
```

```
print("R²   :", r2)
```

Baseline ini sederhana, tetapi sangat berguna sebagai pembanding.

16.9 Latih model

Setelah baseline ada, kita bisa melatih model pertama. Karena ini workflow umum, model yang cocok adalah **LinearRegression**. Referensi lampiran berkali-kali memberi contoh alur yang sama: baca data, split, fit `LinearRegression`, prediksi, evaluasi.

Source code Python lengkap

Simpan sebagai:

```
src/bab16_latih_model.py

import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LinearRegression

df = pd.read_csv("data/raw/workflow_ml_mahasiswa.csv")

X = df[["jam_belajar", "kehadiran", "tugas"]]
y = df["nilai"]

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

model = LinearRegression()
model.fit(X_train, y_train)

print("Intercept:", model.intercept_)
print("Koefisien:")
for fitur, coef in zip(X.columns, model.coef_):
    print(f"{fitur}: {coef}")
```

Apa yang dipelajari

Di tahap ini, pembaca belajar bahwa training berarti:

- model membaca pola dari data train
- model menyusun parameter internal
- hasilnya adalah model terlatih yang siap dipakai prediksi

16.10 Evaluasi

Setelah model dilatih, kita perlu mengevaluasinya pada data uji. Referensi lampiran menegaskan bahwa testing adalah tahap kritis untuk melihat apakah model mampu *generalize*. Untuk regresi, metrik umum yang sangat sering dipakai adalah **MSE**, **RMSE**, dan **R^2**.

Source code Python lengkap

Simpan sebagai:

src/bab16_evaluasi_model.py

```
import pandas as pd
import math
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LinearRegression
from sklearn.metrics import mean_absolute_error,
mean_squared_error, r2_score

df = pd.read_csv("data/raw/workflow_ml_mahasiswa.csv")

X = df[["jam_belajar", "kehadiran", "tugas"]]
y = df["nilai"]

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

model = LinearRegression()
model.fit(X_train, y_train)

y_pred = model.predict(X_test)

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

print("MAE  :", mae)
print("MSE  :", mse)
print("RMSE :", rmse)
print("R²   :", r2)

hasil = pd.DataFrame({
    "aktual": y_test.values,
    "prediksi": y_pred
})
print("\nHasil prediksi:")
print(hasil)
```

Apa yang harus dibaca

- **MAE** → rata-rata error absolut
- **MSE** → error kuadrat, lebih sensitif terhadap error besar
- **RMSE** → akar MSE, kembali ke satuan asli target
- **R^2** → seberapa besar variasi target dijelaskan model

16.11 Iterasi

Ini salah satu bagian terpenting dari workflow. Referensi *AI with Python* menegaskan bahwa pengembangan model yang efektif bersifat **iteratif**: kita menguji, melihat error, lalu memperbaiki fitur, model, atau parameter. Referensi *AI Agents with Python* juga menekankan dokumentasi hasil eksperimen agar pelajaran dari tiap iterasi tidak hilang.

Apa arti iterasi

Setelah evaluasi, kita bertanya:

- apakah error masih terlalu besar?
- apakah ada fitur lain yang lebih relevan?
- apakah scaling diperlukan?
- apakah model lain perlu dicoba?
- apakah baseline sudah berhasil dilampaui cukup jauh?

Contoh iterasi sederhana

Kita bisa coba dua model:

- baseline mean
- linear regression

Lalu bandingkan.

Source code Python lengkap: baseline vs model

Simpan sebagai:

```
src/bab16_iterasi_baseline_vs_model.py

import pandas as pd
import math
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LinearRegression
from sklearn.metrics import mean_absolute_error,
mean_squared_error, r2_score

df = pd.read_csv("data/raw/workflow_ml_mahasiswa.csv")

X = df[["jam_belajar", "kehadiran", "tugas"]]
y = df["nilai"]
```

```
X_train, X_test, y_train, y_test = train_test_split(
    X, y,
    test_size=0.2,
    random_state=42
)

# Baseline
baseline_pred = [y_train.mean()] * len(y_test)
baseline_rmse = math.sqrt(mean_squared_error(y_test,
baseline_pred))
baseline_r2 = r2_score(y_test, baseline_pred)

# Model Linear Regression
model = LinearRegression()
model.fit(X_train, y_train)
y_pred = model.predict(X_test)

model_rmse = math.sqrt(mean_squared_error(y_test, y_pred))
model_r2 = r2_score(y_test, y_pred)

print("=== PERBANDINGAN ===")
print("Baseline RMSE:", baseline_rmse)
print("Baseline R²  :", baseline_r2)
print("Model RMSE   :", model_rmse)
print("Model R²     :", model_r2)
```

Makna iterasi

> Kalau model belum jauh lebih baik dari baseline, kita belum selesai. Itulah tanda bahwa workflow ML bukan “sekali jalan”, tetapi siklus belajar.

16.12 Simpan model sederhana

Setelah model dianggap layak sebagai versi awal, kita bisa menyimpannya. Referensi lampiran tentang deployment dan proyek AI secara eksplisit mencontohkan penyimpanan model dengan **joblib** ke file `.pkl`.

Kenapa model disimpan

> Karena kita tidak ingin melatih ulang dari nol setiap kali ingin memakai model.

Source code Python lengkap

Simpan sebagai:

src/bab16_simpan_model.py

```
import pandas as pd
import joblib
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LinearRegression

df = pd.read_csv("data/raw/workflow_ml_mahasiswa.csv")

X = df[["jam_belajar", "kehadiran", "tugas"]]
y = df["nilai"]

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

model = LinearRegression()
model.fit(X_train, y_train)

joblib.dump(model, "outputs/model_nilai_mahasiswa.pkl")

print("Model berhasil disimpan ke outputs/model_nilai_mahasiswa.pkl")
```

Memuat kembali model

```
import joblib
import pandas as pd

model = joblib.load("outputs/model_nilai_mahasiswa.pkl")

data_baru = pd.DataFrame({
    "jam_belajar": [8],
    "kehadiran": [88],
    "tugas": [84]
})

prediksi = model.predict(data_baru)

print("Prediksi nilai mahasiswa baru:", prediksi[0])
```

16.13 Source code Python lengkap: workflow *end-to-end*

Berikut versi terpadu dari seluruh bab.

Simpan sebagai:

```
src/bab16_workflow_ml_end_to_end.py

import pandas as pd
import math
import joblib
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LinearRegression
from sklearn.metrics import mean_absolute_error, mean_squared_error,
r2_score

print("=" * 80)
print("BAB 16 - WORKFLOW MACHINE LEARNING END-TO-END")
print("=" * 80)

# ============================================================
# 1. DEFINISIKAN MASALAH
# ============================================================
print("\n[1] MASALAH")
print("Memprediksi nilai mahasiswa berdasarkan jam belajar, kehadiran, dan
tugas.")

# ============================================================
# 2. LOAD DATA DAN TENTUKAN TARGET
# ============================================================
df = pd.read_csv("data/raw/workflow_ml_mahasiswa.csv")

print("\n[2] DATA AWAL")
print(df.head())

target = "nilai"
features = ["jam_belajar", "kehadiran", "tugas"]

X = df[features]
y = df[target]

print("\nTarget :", target)
print("Features:", features)

# ============================================================
# 3. SPLIT DATA
# ============================================================
X_train, X_test, y_train, y_test = train_test_split(
    X, y,
    test_size=0.2,
    random_state=42
)

print("\n[3] UKURAN DATA")
print("X_train:", X_train.shape)
print("X_test :", X_test.shape)

# ============================================================
```

```
# 4. BASELINE
# ============================================================
baseline_pred = [y_train.mean()] * len(y_test)

baseline_mae = mean_absolute_error(y_test, baseline_pred)
baseline_mse = mean_squared_error(y_test, baseline_pred)
baseline_rmse = math.sqrt(baseline_mse)
baseline_r2 = r2_score(y_test, baseline_pred)

print("\n[4] BASELINE MODEL")
print("MAE  :", baseline_mae)
print("MSE  :", baseline_mse)
print("RMSE :", baseline_rmse)
print("R²   :", baseline_r2)

# ============================================================
# 5. LATIH MODEL
# ============================================================
model = LinearRegression()
model.fit(X_train, y_train)

print("\n[5] PARAMETER MODEL")
print("Intercept:", model.intercept_)
for fitur, coef in zip(features, model.coef_):
    print(f"{fitur}: {coef}")

# ============================================================
# 6. EVALUASI
# ============================================================
y_pred = model.predict(X_test)

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

print("\n[6] EVALUASI MODEL")
print("MAE  :", mae)
print("MSE  :", mse)
print("RMSE :", rmse)
print("R²   :", r2)

hasil = pd.DataFrame({
    "aktual": y_test.values,
    "prediksi": y_pred
})

print("\nHasil prediksi:")
print(hasil)

# ============================================================
# 7. ITERASI SEDERHANA
# ============================================================
print("\n[7] ITERASI")
if rmse < baseline_rmse:
    print("Model lebih baik daripada baseline. Lanjutkan penyempurnaan.")
else:
    print("Model belum mengalahkan baseline dengan baik. Perlu iterasi
fitur/model.")
```

```
# ============================================================
# 8. SIMPAN MODEL
# ============================================================
joblib.dump(model, "outputs/model_workflow_ml.pkl")
print("\n[8] MODEL DISIMPAN")
print("outputs/model_workflow_ml.pkl")

print("\nSELESAI. Anda sudah menjalankan workflow ML end-to-end.")
print("=" * 80)
```

16.14 Dokumentasi eksperimen

Walau user tidak menulisnya lagi di permintaan terakhir, outline utama menyebut **dokumentasi eksperimen** sebagai bagian workflow yang benar. Ini juga didukung referensi yang menekankan pentingnya catatan hasil eksperimen agar proses tetap transparan dan dapat diulang.

Hal yang sebaiknya dicatat

Untuk setiap eksperimen, biasakan catat:

- nama dataset
- target
- fitur yang dipakai
- cara split
- preprocessing
- model
- parameter
- metrik hasil
- insight singkat
- keputusan lanjutannya

Ini penting agar pembaca tidak “merasa” model membaik tanpa bukti yang tercatat.

16.15 Kesalahan umum pemula

Ada beberapa jebakan yang sangat sering terjadi.

- **langsung memilih algoritma tanpa mendefinisikan masalah**
- **target tidak jelas**
- **fitur dipilih sembarangan**
- **tidak membuat baseline**
- **tidak memisahkan train dan test**
- **evaluasi hanya sekali lalu langsung puas**
- **tidak menyimpan model atau hasil eksperimen**
- **tidak mendokumentasikan iterasi**

Kebiasaan yang lebih sehat adalah selalu bertanya:

> **apakah saya sedang menjalankan model, atau sedang membangun workflow yang bisa dipercaya?**

16.16 Refleksi: ML yang baik adalah proses, bukan trik

Bab ini seharusnya membantu pembaca memahami satu hal besar: *machine learning* yang baik tidak bergantung pada “algoritma paling keren”, tetapi pada **proses yang sehat dari awal sampai akhir**. Referensi lampiran juga konsisten menunjukkan bahwa proyek ML yang baik selalu bergerak melalui tahapan yang jelas: data, preprocessing, split, training, testing, evaluasi, iterasi, dan kadang penyimpanan/deployment model.

Itulah mengapa workflow *end-to-end* ini penting. Begitu pembaca memahami alurnya, masuk ke Logistic Regression, KNN, Decision Tree, atau Random Forest di bab-bab berikutnya akan terasa jauh lebih masuk akal. Algoritma akan berubah, tetapi workflow intinya relatif tetap.

Penutup Bab

Bab ini merangkum **workflow machine learning end-to-end** yang sehat: **definisikan masalah, tentukan target, pilih feature, split data, bangun baseline, latih model, evaluasi, iterasi, dan simpan model sederhana**. Ini sangat selaras dengan outline naskah utama dan didukung kuat oleh materi lampiran yang menempatkan preprocessing, train-test split, evaluasi, iterasi, dan dokumentasi sebagai fondasi proyek ML yang baik.

Pesan terpenting dari bab ini: model yang bagus lahir dari **workflow yang disiplin**, bukan dari keberuntungan saat menjalankan satu algoritma.

Tantangan kecil untuk pembaca: modifikasi file `bab16_workflow_ml_end_to_end.py` agar selain `LinearRegression`, Anda juga mencoba satu baseline lain yang lebih cerdas, misalnya hanya memakai satu fitur `jam_belajar`, lalu bandingkan metriknya dengan model yang memakai tiga fitur.

Bab 17. Bagaimana Memilih Algoritma yang Cocok?

Bab ini membantu pembaca menjawab pertanyaan yang sangat sering muncul setelah memahami workflow ML: **“kalau datanya sudah siap, saya harus mulai dari algoritma yang mana?”** Di outline naskah utama, bagian ini memang diarahkan untuk memberi panduan praktis: jika target berupa angka maka pilih keluarga regresi, jika target berupa kelas maka pilih klasifikasi, jika tidak ada label maka masuk ke clustering, lalu bandingkan algoritma dari sisi **interpretabilitas, akurasi, kompleksitas, dan kebutuhan preprocessing**. Pendekatan seperti ini sangat cocok untuk pembaca pemula karena fokusnya bukan pada hafalan nama algoritma, tetapi pada **cara memilih dengan logis**.

Materi lampiran juga cukup konsisten. Referensi Jain membedakan **regression** untuk target kontinu, **classification** untuk target kategorikal, dan **clustering** sebagai teknik *unsupervised learning* untuk mengelompokkan data tanpa label. Referensi lain menekankan bahwa **Linear Regression**, **Logistic Regression**, dan **Decision Tree** relatif mudah dijelaskan; **KNN** dan **Decision Tree** cocok sebagai baseline cepat; **Random Forest** memberi peningkatan akurasi dan ketahanan; **Gradient Boosting** kuat untuk data tabular; dan **K-Means** umum dipakai untuk segmentasi. Di sisi clustering, beberapa referensi juga mengingatkan bahwa pilihan algoritma harus mempertimbangkan bentuk data, outlier, dan tujuan analisis.

Bab ini dirancang agar pembaca mampu:

- menentukan keluarga algoritma dari bentuk masalahnya
- memilih model awal yang masuk akal
- memahami kapan perlu model sederhana dan kapan perlu model lebih kuat
- membaca kompromi antara **mudah dijelaskan**, **akurasi**, **kompleksitas**, dan **preprocessing**
- mencoba **source code Python lengkap** untuk beberapa algoritma awal

17.1 Mulai dari pertanyaan, bukan dari nama algoritma

Kesalahan paling umum pemula adalah memulai dari pertanyaan seperti:

- “pakai Random Forest atau XGBoost?”
- “KNN bagus tidak?”
- “Decision Tree akurat tidak?”

Padahal pertanyaan yang lebih sehat adalah:

- target saya **angka** atau **kelas**?
- apakah saya punya **label**?
- apakah saya perlu model yang **mudah dijelaskan**?
- apakah saya sedang membuat **baseline cepat**?
- apakah data saya **tabular sederhana** atau **lebih kompleks**?
- apakah saya sedang mencari **segmentasi**?

Dengan cara berpikir ini, pemilihan algoritma menjadi jauh lebih masuk akal.

17.2 Jika target berupa angka → regresi

Kalau target yang ingin diprediksi berupa **nilai numerik kontinu**, maka keluarga masalahnya adalah **regresi**. Referensi Jain menjelaskan bahwa regresi dipakai ketika target berupa nilai kontinu, dengan contoh seperti harga rumah dan temperatur hari berikutnya. Ia juga menempatkan **Linear Regression**, **Decision Tree for Regression**, dan **Support Vector Regression** sebagai contoh algoritma utama di keluarga ini.

Contoh masalah regresi

- prediksi harga rumah
- prediksi omzet
- prediksi konsumsi energi
- prediksi nilai mahasiswa
- prediksi biaya operasional

Algoritma awal yang masuk akal

- **Linear Regression** → paling mudah dipahami
- **Decision Tree Regressor** → mudah divisualisasikan, bisa menangkap nonlinieritas sederhana
- **Random Forest Regressor** → lebih kuat untuk data tabular dengan pola lebih kompleks
- **Gradient Boosting Regressor** → sering sangat kuat pada data tabular

Kapan mulai dari Linear Regression

Mulailah dari **Linear Regression** jika:

- Anda ingin **model mudah dijelaskan**
- hubungan antarvariabel tampak cukup linear
- Anda butuh baseline interpretatif
- Anda ingin membaca koefisien dan pengaruh fitur

Source code Python lengkap: regresi sederhana

Simpan sebagai:

```
src/bab17_pilih_algoritma_regresi.py

import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LinearRegression
from sklearn.metrics import mean_absolute_error, mean_squared_error,
r2_score
import math

# Dataset regresi sederhana
df = pd.DataFrame({
    "jam_belajar": [2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12],
```

```
    "kehadiran": [70, 75, 78, 80, 82, 85, 88, 90, 92, 95, 96],
    "nilai": [58, 63, 67, 71, 75, 79, 84, 88, 91, 94, 97]
})

X = df[["jam_belajar", "kehadiran"]]
y = df["nilai"]

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

model = LinearRegression()
model.fit(X_train, y_train)

y_pred = model.predict(X_test)

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

print("=== REGRESI ===")
print("Intercept:", model.intercept_)
print("Koefisien:")
for fitur, coef in zip(X.columns, model.coef_):
    print(f"{fitur}: {coef}")

print("\nMAE :", mae)
print("MSE :", mse)
print("RMSE:", rmse)
print("R²  :", r2)
```

17.3 Jika target berupa kelas → klasifikasi

Kalau target berupa **kategori**, maka keluarga masalahnya adalah **klasifikasi**. Jain menegaskan bahwa klasifikasi dipakai ketika target bersifat kategorikal, misalnya spam/tidak spam atau churn/tidak churn. Ia menempatkan **Logistic Regression**, **Decision Tree for Classification**, dan **KNN** sebagai algoritma inti untuk tahap awal.

Contoh masalah klasifikasi

- lulus / tidak lulus
- fraud / tidak fraud
- spam / tidak spam
- churn / tidak churn
- risiko rendah / sedang / tinggi

Algoritma awal yang masuk akal

- **Logistic Regression** → sederhana, kuat, dan cukup interpretatif
- **Decision Tree Classifier** → mudah dijelaskan dan divisualisasikan

- **KNN** → baseline cepat
- **Random Forest Classifier** → sering lebih stabil dan lebih akurat
- **Gradient Boosting Classifier** → sangat kuat pada data tabular

Kapan mulai dari Logistic Regression

Mulailah dari **Logistic Regression** jika:

- target biner atau kelas sederhana
- Anda ingin baseline yang kuat dan mudah dijelaskan
- Anda ingin probabilitas kelas, bukan hanya label
- Anda ingin model awal yang sering cukup kompetitif

Referensi lampiran juga menekankan bahwa Logistic Regression adalah model penting yang menyeimbangkan **simplicity, interpretability, and efficacy** pada klasifikasi biner.

Source code Python lengkap: klasifikasi sederhana

Simpan sebagai:

```
src/bab17_pilih_algoritma_klasifikasi.py

import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import accuracy_score, classification_report,
confusion_matrix

# Dataset klasifikasi sederhana
df = pd.DataFrame({
    "jam_belajar": [2, 3, 4, 5, 6, 7, 8, 9, 10, 11],
    "kehadiran":   [70, 72, 75, 78, 80, 85, 88, 90, 92, 95],
    "lulus":       [0, 0, 0, 0, 1, 1, 1, 1, 1, 1]
})

X = df[["jam_belajar", "kehadiran"]]
y = df["lulus"]

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

model = LogisticRegression()
model.fit(X_train, y_train)

y_pred = model.predict(X_test)

print("=== KLASIFIKASI ===")
print("Accuracy:", accuracy_score(y_test, y_pred))
print("\nConfusion Matrix:")
print(confusion_matrix(y_test, y_pred))
```

```
print("\nClassification Report:")
print(classification_report(y_test, y_pred))
```

17.4 Jika tidak ada label → clustering

Kalau data **tidak punya label target**, dan kita ingin mencari pola kelompok alami, maka kita masuk ke wilayah **unsupervised learning**, khususnya **clustering**. Referensi Knox dan Uddin menjelaskan clustering sebagai masalah tanpa label, di mana algoritma mengelompokkan data berdasarkan kemiripan internal. K-Means adalah salah satu algoritma yang paling populer dan cepat untuk tujuan ini.

Contoh kasus clustering

- segmentasi pelanggan
- pengelompokan siswa berdasarkan pola belajar
- segmentasi produk
- pengelompokan wilayah berdasarkan profil transaksi

Kapan K-Means cocok

- Anda ingin segmentasi cepat
- data numerik cukup bersih
- cluster diperkirakan berbentuk cukup kompak
- Anda bisa menentukan jumlah cluster k

Catatan penting tentang K-Means

Referensi Uddin menegaskan bahwa K-Means sensitif terhadap:

- pemilihan centroid awal
- outlier
- penentuan jumlah cluster
- bentuk cluster yang tidak ideal untuk Euclidean centroid-based clustering

Source code Python lengkap: clustering dengan K-Means

Simpan sebagai:

```
src/bab17_pilih_algoritma_clustering.py

import pandas as pd
from sklearn.cluster import KMeans

# Dataset segmentasi pelanggan sederhana
df = pd.DataFrame({
    "belanja_bulanan": [1, 2, 2, 3, 8, 9, 10, 11],
    "frekuensi_kunjungan": [1, 1, 2, 2, 8, 9, 9, 10]
})

X = df[["belanja_bulanan", "frekuensi_kunjungan"]]

model = KMeans(n_clusters=2, random_state=42, n_init=10)
df["cluster"] = model.fit_predict(X)

print("=== CLUSTERING ===")
print(df)
print("\nCentroid:")
print(model.cluster_centers_)
```

17.5 Jika butuh model mudah dijelaskan

Ada banyak situasi di mana **interpretabilitas** lebih penting daripada akurasi maksimum. Misalnya:

- pendidikan
- kebijakan publik
- audit
- kesehatan
- presentasi ke pimpinan nonteknis

Dalam kasus seperti ini, referensi lampiran sangat mendukung pilihan model yang mudah dijelaskan, seperti:

- **Linear Regression**
- **Logistic Regression**
- **Decision Tree**

Kenapa model-model ini mudah dijelaskan

- **Linear Regression** → bisa membaca koefisien
- **Logistic Regression** → bisa membaca arah pengaruh fitur dan probabilitas
- **Decision Tree** → bisa divisualisasikan sebagai aturan keputusan

Source code Python lengkap: Decision Tree yang mudah dijelaskan

Simpan sebagai:

```
src/bab17_model_mudah_dijelaskan.py
```

```
import pandas as pd
from sklearn.tree import DecisionTreeClassifier, export_text

df = pd.DataFrame({
    "jam_belajar": [2, 3, 4, 5, 6, 7, 8, 9],
    "kehadiran":   [70, 72, 75, 78, 82, 85, 88, 90],
    "lulus":       [0, 0, 0, 0, 1, 1, 1, 1]
})

X = df[["jam_belajar", "kehadiran"]]
y = df["lulus"]

model = DecisionTreeClassifier(max_depth=3, random_state=42)
model.fit(X, y)

aturan = export_text(model, feature_names=list(X.columns))

print("=== DECISION TREE RULES ===")
print(aturan)
```

17.6 Jika ingin baseline cepat

Kalau target Anda adalah **memulai cepat** dan mendapat pembanding awal, maka pilihan yang sering masuk akal adalah:

- **KNN**
- **Decision Tree**

Referensi Jain memang mencantumkan **KNN** dan **Decision Tree** sebagai algoritma inti yang sederhana untuk klasifikasi, sementara Decision Tree juga bisa dipakai untuk regresi. KNN sering jadi baseline cepat karena tinggal memilih k, sedangkan Decision Tree cepat memberi model yang bisa langsung jalan.

KNN cocok jika:

- dataset tidak terlalu besar
- baseline cepat dibutuhkan
- fitur numerik sudah di-*scale*
- Anda ingin model yang simpel secara konsep

Decision Tree cocok jika:

- ingin baseline cepat
- data campuran cukup mudah dipisahkan aturan

- ingin hasil yang bisa divisualisasikan
- tidak ingin terlalu repot scaling di tahap awal

Source code Python lengkap: baseline KNN

Simpan sebagai:

```
src/bab17_baseline_knn.py

import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.neighbors import KNeighborsClassifier
from sklearn.metrics import accuracy_score

df = pd.DataFrame({
    "jam_belajar": [2, 3, 4, 5, 6, 7, 8, 9, 10, 11],
    "kehadiran":   [70, 72, 75, 78, 80, 85, 88, 90, 92, 95],
    "lulus":       [0, 0, 0, 0, 1, 1, 1, 1, 1, 1]
})

X = df[["jam_belajar", "kehadiran"]]
y = df["lulus"]

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

scaler = StandardScaler()
X_train_scaled = scaler.fit_transform(X_train)
X_test_scaled = scaler.transform(X_test)

model = KNeighborsClassifier(n_neighbors=3)
model.fit(X_train_scaled, y_train)

y_pred = model.predict(X_test_scaled)

print("=== KNN BASELINE ===")
print("Accuracy:", accuracy_score(y_test, y_pred))
```

17.7 Jika data tabular cukup kompleks

Untuk data tabular nyata yang polanya tidak terlalu sederhana, seringkali model *tree-based ensemble* bekerja sangat baik. Materi lampiran menekankan bahwa:

- **Random Forest** meningkatkan akurasi dan robustness dengan menggabungkan banyak decision tree
- **Gradient Boosting** sangat kuat untuk data tabular dan menjadi dasar banyak model modern seperti XGBoost/LightGBM/CatBoost

Kapan Random Forest cocok

- data tabular punya pola nonlinier
- ingin model kuat tanpa banyak *hyperparameter* rumit di awal
- ingin baseline menengah yang sering lebih stabil dari tree tunggal
- data tidak terlalu besar untuk memori

Kapan Gradient Boosting cocok

- ingin akurasi tinggi pada tabular data
- siap menerima model yang lebih kompleks
- siap melakukan evaluasi dan tuning lebih lanjut

Catatan praktis

- **Random Forest** biasanya lebih mudah sebagai langkah awal
- **Gradient Boosting** sering lebih kuat, tetapi lebih kompleks
- untuk buku pemula, cukup pahami bahwa keduanya sering jadi andalan pada **tabular data**

Source code Python lengkap: Random Forest sederhana

Simpan sebagai:

```
src/bab17_random_forest_tabular.py

import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import accuracy_score

df = pd.DataFrame({
    "jam_belajar": [2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13],
    "kehadiran":   [70, 72, 75, 78, 80, 85, 88, 90, 92, 95, 96, 97],
    "tugas":       [60, 62, 65, 68, 72, 76, 80, 84, 87, 90, 92, 95],
    "lulus":       [0, 0, 0, 0, 1, 1, 1, 1, 1, 1, 1, 1]
})

X = df[["jam_belajar", "kehadiran", "tugas"]]
y = df["lulus"]

X_train, X_test, y_train, y_test = train_test_split(
    X, y, test_size=0.25, random_state=42
)

model = RandomForestClassifier(random_state=42)
model.fit(X_train, y_train)

y_pred = model.predict(X_test)

print("=== RANDOM FOREST ===")
```

```
print("Accuracy:", accuracy_score(y_test, y_pred))
```

17.8 Jika ingin segmentasi → K-Means

Kalau tujuan utamanya adalah **segmentasi**, maka **K-Means** hampir selalu menjadi titik awal yang masuk akal. Referensi lampiran berulang kali menyebut K-Means sebagai algoritma clustering populer untuk segmentasi pelanggan, karena cepat, sederhana, dan mudah divisualisasikan.

Cocok untuk:

- segmentasi pelanggan
- segmentasi mahasiswa
- segmentasi produk
- clustering awal untuk eksplorasi

Tetapi ingat

> K-Means bukan solusi untuk semua bentuk cluster. Jika data penuh outlier atau cluster tidak bulat/kompak, algoritma lain seperti DBSCAN atau Agglomerative bisa lebih cocok. Itu juga disinggung dalam referensi Uddin.

17.9 Perbandingan sederhana: interpretabilitas, akurasi, kompleksitas, kebutuhan preprocessing

Bagian ini adalah inti praktis bab. Berikut cara membaca kompromi antar algoritma.

1. Linear Regression / Logistic Regression

> **Kelebihan:**
>
> - mudah dijelaskan
> - cepat
> - baseline kuat
> - koefisien bisa dibaca
>
> **Kekurangan:**
>
> - asumsi linear
> - bisa kurang kuat pada pola kompleks
> - sensitif pada multikolinearitas, terutama logistic regression menurut referensi Van Der Post
>
> **Preprocessing:**
>
> - biasanya lebih baik jika fitur numerik rapi
> - scaling sering membantu, terutama logistic

2. Decision Tree

Kelebihan:

- sangat mudah dijelaskan
- tidak terlalu menuntut scaling
- bisa menangkap nonlinieritas

Kekurangan:

- mudah overfit
- bisa tidak stabil terhadap perubahan data kecil

Preprocessing:

- relatif ringan
- cocok untuk baseline cepat

3. KNN

Kelebihan:

- konsep sederhana
- baseline cepat
- tidak perlu training rumit

Kekurangan:

- sensitif terhadap scaling
- lambat saat prediksi jika data besar
- sensitif terhadap pemilihan k

Preprocessing:

- scaling sangat penting

4. Random Forest

Kelebihan:

- kuat untuk tabular data
- lebih robust daripada tree tunggal
- sering jadi pilihan awal yang aman

Kekurangan:

- lebih sulit dijelaskan daripada tree tunggal
- model lebih berat

Preprocessing:

- umumnya tidak terlalu menuntut scaling

5. Gradient Boosting

Kelebihan:

- sering sangat akurat untuk tabular data
- kuat pada pola kompleks

Kekurangan:

- lebih kompleks
- tuning dan interpretasi lebih menantang
- bisa lebih sensitif terhadap setting

Preprocessing:

- tetap butuh data yang rapi, meski scaling tidak selalu sepenting KNN/SVM

6. K-Means

Kelebihan:

- cepat
- mudah divisualisasikan
- bagus untuk segmentasi awal

Kekurangan:

- harus menentukan jumlah cluster
- sensitif terhadap outlier
- kurang cocok untuk cluster berbentuk aneh

Preprocessing:

- scaling biasanya penting
- data numerik lebih ideal

17.10 Aturan cepat memilih algoritma

Agar pembaca mudah mengingat, berikut aturan praktis yang bisa dipakai sebagai *mental checklist*:

- **Jika target berupa angka → mulai dari regresi**
    - Linear Regression dulu
    - lalu Random Forest / Gradient Boosting jika perlu
- **Jika target berupa kelas → mulai dari klasifikasi**
    - Logistic Regression atau Decision Tree dulu
    - lalu KNN / Random Forest / Gradient Boosting jika perlu
- **Jika tidak ada label → mulai dari clustering**
    - K-Means untuk segmentasi awal
    - pertimbangkan algoritma lain jika data lebih rumit
- **Jika butuh model mudah dijelaskan**
    - Linear Regression
    - Logistic Regression
    - Decision Tree
- **Jika ingin baseline cepat**
    - KNN
    - Decision Tree
- **Jika data tabular cukup kompleks**
    - Random Forest
    - Gradient Boosting
- **Jika tujuan utama segmentasi**
    - K-Means terlebih dahulu

Aturan ini sejalan dengan semangat materi lampiran yang membedakan algoritma bukan sebagai “siapa paling hebat”, tetapi sebagai **alat untuk tipe masalah tertentu**.

17.11 Source code Python lengkap: memilih algoritma dari jenis masalah

Simpan sebagai:

src/bab17_pilih_algoritma_dari_masalah.py

```
def rekomendasi_algoritma(jenis_target, perlu_interpretasi=False,
baseline_cepat=False, data_tabular_kompleks=False,
butuh_segmentasi=False):
    if butuh_segmentasi:
        return "K-Means"

    if jenis_target == "angka":
        if perlu_interpretasi:
            return "Linear Regression"
        if data_tabular_kompleks:
            return "Random Forest / Gradient Boosting"
        return "Linear Regression"

    if jenis_target == "kelas":
        if perlu_interpretasi:
            return "Logistic Regression / Decision Tree"
        if baseline_cepat:
            return "KNN / Decision Tree"
        if data_tabular_kompleks:
            return "Random Forest / Gradient Boosting"
        return "Logistic Regression"

    if jenis_target == "tanpa_label":
        return "K-Means"

    return "Perlu definisi masalah lebih jelas"

# Contoh penggunaan
print("Kasus 1:", rekomendasi_algoritma("angka"))
print("Kasus 2:", rekomendasi_algoritma("kelas",
perlu_interpretasi=True))
print("Kasus 3:", rekomendasi_algoritma("kelas", baseline_cepat=True))
print("Kasus 4:", rekomendasi_algoritma("kelas",
data_tabular_kompleks=True))
print("Kasus 5:", rekomendasi_algoritma("tanpa_label",
butuh_segmentasi=True))
```

Program ini memang sederhana, tetapi bagus sebagai alat bantu berpikir bagi pemula.

17.12 Refleksi: memilih algoritma adalah soal kompromi, bukan dogma

Pelajaran terpenting dari bab ini adalah bahwa **tidak ada satu algoritma terbaik untuk semua kasus**. Referensi clustering bahkan menegaskan bahwa algoritma yang berbeda bisa unggul pada aspek yang berbeda, tergantung karakter data dan tujuan analisis. Hal yang sama berlaku lebih luas pada ML tabular: model yang paling akurat belum tentu paling mudah dijelaskan, dan model yang paling mudah dijelaskan belum tentu paling kuat.

Karena itu, memilih algoritma selalu berarti mempertimbangkan kompromi antara:

- **interpretabilitas**
- **akurasi**
- **kompleksitas**
- **kebutuhan preprocessing**
- **tujuan analisis**

Inilah pola pikir yang sehat untuk dibangun sejak awal.

Penutup Bab

Bab ini membantu pembaca memilih algoritma secara lebih tenang dan logis. Jika target berupa angka, kita masuk ke **regresi**. Jika target berupa kelas, kita masuk ke **klasifikasi**. Jika tidak ada label, kita masuk ke **clustering**. Jika butuh model mudah dijelaskan, pilih **Linear Regression**, **Logistic Regression**, atau **Decision Tree**. Jika ingin baseline cepat, mulai dari **KNN** atau **Decision Tree**. Jika data tabular cukup kompleks, pertimbangkan **Random Forest** atau **Gradient Boosting**. Jika tujuan utama adalah segmentasi, mulai dari **K-Means**. Semua pilihan ini konsisten dengan materi lampiran tentang supervised vs unsupervised learning, regression vs classification, algoritma dasar klasifikasi, ensemble tabular models, dan clustering untuk segmentasi.

Pesan terpenting dari bab ini: pilih algoritma dari **bentuk masalah dan tujuan penggunaan**, bukan dari tren atau nama yang terdengar paling canggih.

Tantangan kecil untuk pembaca: buat satu skrip baru yang melatih tiga model pada dataset klasifikasi sederhana yang sama — **Logistic Regression**, **Decision Tree**, dan **Random Forest** — lalu bandingkan mana yang paling mudah dijelaskan dan mana yang memberi hasil paling baik.

Bab 18. Contoh Pemilihan Algoritma + Source Code

Bab ini dibuat untuk menjawab kebutuhan yang sangat praktis: **setelah memahami cara memilih algoritma secara konseptual, seperti apa contoh penerapannya di kode?** Di outline buku utama, arah bab ini sangat jelas: tampilkan beberapa kasus umum, lalu pasangkan dengan algoritma yang masuk akal. Itu sangat selaras dengan materi lampiran yang membahas **choosing the right algorithm**, **model selection**, **complexity vs interpretability**, serta perbedaan antara **supervised** dan **unsupervised learning**. Referensi lampiran juga berulang kali menunjukkan pola pemilihan yang serupa: **prediksi harga rumah** cocok dimulai dari *Linear Regression*, **kasus biner** sering cocok dimulai dari *Logistic Regression*, **klasifikasi tabular yang lebih kaya** sering diuntungkan oleh *Random Forest*, dan **segmentasi tanpa label** sangat cocok diawali dengan *K-Means*.

Bab ini sengaja dibuat **langsung pakai**, dengan empat kasus yang sangat mudah dipahami:

- **Kasus prediksi harga rumah** → `LinearRegression`
- **Kasus lulus/tidak lulus** → `LogisticRegression`
- **Kasus klasifikasi pelanggan** → `RandomForestClassifier`
- **Kasus segmentasi pelanggan** → `KMeans`

Di akhir bab, saya sertakan juga **template kode umum** untuk memilih model, melakukan `fit`, `predict`, dan `evaluate`, agar pembaca bisa lebih mudah mengulang pola yang sama pada dataset lain. Pendekatan seperti ini juga sejalan dengan referensi yang menekankan pentingnya membangun *workflow* yang konsisten di scikit-learn.

18.1 Cara membaca bab ini

Sebelum masuk ke kode, ada satu prinsip yang perlu dipegang:

kita tidak memilih algoritma karena sedang populer, tetapi karena cocok dengan bentuk masalahnya.

Cara berpikir sederhananya adalah:

- kalau target berupa **angka** → pikirkan **regresi**
- kalau target berupa **kelas** → pikirkan **klasifikasi**
- kalau **tidak ada label** → pikirkan **clustering**
- kalau butuh model yang **mudah dijelaskan** → mulai dari model yang sederhana
- kalau data tabular cukup kaya dan pola lebih kompleks → pertimbangkan model ensemble seperti *Random Forest* atau *Gradient Boosting*

Bab ini bukan berarti “algoritma ini selalu terbaik”, tetapi memberi **titik mulai yang sehat** untuk pemula.

18.2 Kasus 1 — Prediksi harga rumah → `LinearRegression`

Mengapa LinearRegression cocok

Kalau target yang ingin diprediksi berupa **harga**, yaitu nilai numerik kontinu, maka keluarga masalahnya adalah **regresi**. Dalam referensi Jain, *Linear Regression* diberikan sebagai contoh paling langsung untuk **predicting house prices**, dengan fitur seperti ukuran rumah dan target berupa harga. Referensi lain juga menempatkan *Linear Regression* sebagai model awal yang baik untuk memprediksi harga rumah karena **mudah dijelaskan**, **mudah divisualisasikan**, dan menjadi baseline yang sangat penting sebelum mencoba model regresi yang lebih kompleks.

Kapan LinearRegression menjadi pilihan bagus

Gunakan `LinearRegression` jika:

- target berupa angka
- Anda ingin baseline yang mudah dipahami
- hubungan antar fitur dan target tampak cukup linear
- interpretasi koefisien penting

Dataset contoh

Simpan sebagai:

```
data/raw/rumah_regresi.csv
```

Isi file:

```
luas_rumah,jumlah_kamar,harga
100,2,300000
120,2,360000
140,3,420000
160,3,500000
180,4,560000
200,4,620000
220,5,700000
240,5,760000
260,6,820000
280,6,900000
```

Source code Python lengkap

Simpan sebagai:

src/bab18_kasus_rumah_linear_regression.py

```
import pandas as pd
import math
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LinearRegression
from sklearn.metrics import mean_absolute_error,
mean_squared_error, r2_score

# 1. Load data
df = pd.read_csv("data/raw/rumah_regresi.csv")

# 2. Pilih fitur dan target
X = df[["luas_rumah", "jumlah_kamar"]]
y = df["harga"]

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

# 4. Bangun dan latih model
model = LinearRegression()
model.fit(X_train, y_train)

# 5. Prediksi
y_pred = model.predict(X_test)

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

print("=== LINEAR REGRESSION UNTUK HARGA RUMAH ===")
print("Intercept:", model.intercept_)
print("Koefisien:")
for fitur, coef in zip(X.columns, model.coef_):
    print(f"{fitur}: {coef}")

print("\nEvaluasi:")
print("MAE  :", mae)
print("MSE  :", mse)
print("RMSE :", rmse)
print("R²   :", r2)

hasil = pd.DataFrame({
    "aktual": y_test.values,
    "prediksi": y_pred
```

```
})

print("\nHasil prediksi:")
print(hasil)
```

Apa yang dipelajari dari kasus ini

Kasus ini mengajarkan bahwa untuk target numerik, kita mulai dari regresi yang mudah dipahami. Bila hasilnya belum cukup baik, baru kita berpikir ke model lain seperti *Ridge*, *Lasso*, atau model ensemble regresi. Referensi lampiran juga menekankan bahwa proyek regresi yang baik dimulai dari preprocessing, training, evaluasi, lalu interpretasi *feature importance* atau koefisien.

18.3 Kasus 2 — Lulus/tidak lulus → `LogisticRegression`

Mengapa LogisticRegression cocok

Kalau target berupa **kelas biner**, misalnya `Lulus` atau `Tidak`, maka kita masuk ke wilayah **klasifikasi**. Referensi lampiran sangat eksplisit bahwa meskipun namanya *Logistic Regression*, model ini dipakai untuk **classification problems**, khususnya **binary outcomes**, dan dipuji karena **straightforward, efficient, and interpretable**.

Kapan LogisticRegression menjadi pilihan bagus

Gunakan `LogisticRegression` jika:

- target berupa kelas biner
- Anda ingin model klasifikasi yang sederhana dan kuat
- probabilitas kelas penting
- interpretasi arah pengaruh fitur penting

Dataset contoh

Simpan sebagai:

```
data/raw/lulus_logistic.csv
```

Isi file:

```
jam_belajar,kehadiran,lulus
2,70,0
3,72,0
4,75,0
5,78,0
6,80,1
7,84,1
8,87,1
9,90,1
10,92,1
11,95,1
```

Source code Python lengkap

Simpan sebagai:

src/bab18_kasus_lulus_logistic_regression.py

```
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import accuracy_score, confusion_matrix,
classification_report

# 1. Load data
df = pd.read_csv("data/raw/lulus_logistic.csv")

# 2. Pilih fitur dan target
X = df[["jam_belajar", "kehadiran"]]
y = df["lulus"]

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

# 4. Scaling
scaler = StandardScaler()
X_train_scaled = scaler.fit_transform(X_train)
X_test_scaled = scaler.transform(X_test)

# 5. Bangun dan latih model
model = LogisticRegression()
model.fit(X_train_scaled, y_train)

# 6. Prediksi
y_pred = model.predict(X_test_scaled)
y_prob = model.predict_proba(X_test_scaled)

# 7. Evaluasi
print("=== LOGISTIC REGRESSION UNTUK LULUS/TIDAK LULUS ===")
print("Accuracy:", accuracy_score(y_test, y_pred))
print("\nConfusion Matrix:")
print(confusion_matrix(y_test, y_pred))
print("\nClassification Report:")
print(classification_report(y_test, y_pred))

hasil = pd.DataFrame({
    "aktual": y_test.values,
    "prediksi": y_pred,
    "prob_tidak_lulus": y_prob[:, 0],
    "prob_lulus": y_prob[:, 1]
})
```

```
print("\nHasil prediksi:")
print(hasil)
```

Apa yang dipelajari dari kasus ini

Kasus ini menunjukkan bahwa ketika target berupa kelas biner, *Logistic Regression* sangat cocok sebagai baseline kuat. Ia juga membantu pembaca melihat bahwa klasifikasi tidak hanya menghasilkan label, tetapi juga **probabilitas kelas**, yang sering sangat penting untuk pengambilan keputusan.

18.4 Kasus 3 — Klasifikasi pelanggan → `RandomForestClassifier`

Mengapa RandomForestClassifier cocok

Ketika data tabular klasifikasi mulai lebih kaya dan pola tidak sesederhana hubungan linear, model ensemble seperti **Random Forest** sering menjadi pilihan awal yang sangat aman. Referensi lampiran menekankan bahwa *Random Forest* membangun banyak *decision tree* dan menggabungkan hasilnya untuk **meningkatkan akurasi dan robustness**, serta mengurangi risiko *overfitting* dibanding pohon tunggal. Ini membuatnya sangat cocok untuk **klasifikasi pelanggan**, **credit scoring**, dan tugas tabular lain yang cukup kompleks.

Kapan RandomForestClassifier menjadi pilihan bagus

Gunakan `RandomForestClassifier` jika:

- target berupa kelas
- data tabular punya beberapa fitur
- pola mungkin nonlinier
- Anda ingin model yang lebih kuat daripada baseline sederhana
- interpretasi total tidak harus sesederhana *Logistic Regression*

Dataset contoh

Simpan sebagai:

```
data/raw/pelanggan_random_forest.csv
```

Isi file:

```
umur,pendapatan,riwayat_belanja,status_pelanggan
22,3,1,0
25,4,2,0
28,5,3,0
30,6,4,1
35,7,5,1
40,8,6,1
45,9,7,1
50,10,8,1
```

```
27,4,2,0
33,6,4,1
38,7,5,1
29,5,3,0
```

Anggap `status_pelanggan` adalah label 0/1, misalnya:

- `0` = pelanggan biasa
- `1` = pelanggan bernilai tinggi

Source code Python lengkap

Simpan sebagai:

```
src/bab18_kasus_pelanggan_random_forest.py

import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import accuracy_score, confusion_matrix,
classification_report

# 1. Load data
df = pd.read_csv("data/raw/pelanggan_random_forest.csv")

# 2. Pilih fitur dan target
X = df[["umur", "pendapatan", "riwayat_belanja"]]
y = df["status_pelanggan"]

# 3. Split data
X_train, X_test, y_train, y_test = train_test_split(
    X, y, test_size=0.25, random_state=42
)

# 4. Bangun dan latih model
model = RandomForestClassifier(random_state=42)
model.fit(X_train, y_train)

# 5. Prediksi
y_pred = model.predict(X_test)

# 6. Evaluasi
print("=== RANDOM FOREST CLASSIFIER UNTUK KLASIFIKASI PELANGGAN ===")
print("Accuracy:", accuracy_score(y_test, y_pred))
print("\nConfusion Matrix:")
print(confusion_matrix(y_test, y_pred))
print("\nClassification Report:")
print(classification_report(y_test, y_pred))

# 7. Feature importance
importances = pd.DataFrame({
    "fitur": X.columns,
    "importance": model.feature_importances_
}).sort_values(by="importance", ascending=False)

print("\nFeature Importance:")
print(importances)
```

Apa yang dipelajari dari kasus ini

Kasus ini mengajarkan bahwa saat klasifikasi tabular menjadi sedikit lebih kompleks, *Random Forest* sering menjadi pilihan masuk akal karena:

- relatif kuat
- cukup tahan terhadap noise
- memberi *feature importance*
- tidak terlalu menuntut scaling dibanding KNN atau SVM

18.5 Kasus 4 — Segmentasi pelanggan → `KMeans`

Mengapa KMeans cocok

Kalau tidak ada label target dan tujuan kita adalah **segmentasi pelanggan**, maka kita masuk ke **clustering**. Referensi lampiran berulang kali menempatkan **K-Means Clustering** sebagai algoritma *unsupervised learning* yang sangat populer untuk **customer segmentation**, karena ia cepat, sederhana, dan mudah dipakai sebagai eksplorasi awal.

Kapan KMeans menjadi pilihan bagus

Gunakan `KMeans` jika:

- tidak ada label target
- tujuan utamanya segmentasi
- fitur utama numerik
- Anda ingin eksplorasi cluster yang cepat
- data relatif cukup kompak dan tidak terlalu aneh bentuknya

Dataset contoh

Simpan sebagai:

```
data/raw/segmentasi_kmeans.csv
```

Isi file:

```
pendapatan_tahunan,skor_belanja
15,39
16,81
17,6
18,77
35,40
36,76
37,6
38,72
60,55
62,52
64,59
```

```
66,54
90,20
92,18
94,22
96,19
```

Source code Python lengkap

Simpan sebagai:

```
src/bab18_kasus_segmentasi_kmeans.py

import pandas as pd
from sklearn.cluster import KMeans
from sklearn.preprocessing import StandardScaler

# 1. Load data
df = pd.read_csv("data/raw/segmentasi_kmeans.csv")

# 2. Pilih fitur
X = df[["pendapatan_tahunan", "skor_belanja"]]

# 3. Scaling
scaler = StandardScaler()
X_scaled = scaler.fit_transform(X)

# 4. Bangun model clustering
model = KMeans(n_clusters=3, random_state=42, n_init=10)
df["cluster"] = model.fit_predict(X_scaled)

# 5. Hasil
print("=== KMEANS UNTUK SEGMENTASI PELANGGAN ===")
print(df)

print("\nCentroid (dalam skala hasil standardisasi):")
print(model.cluster_centers_)
```

Apa yang dipelajari dari kasus ini

Kasus ini menunjukkan bahwa untuk segmentasi awal, K-Means sangat cocok. Ia membantu pembaca melihat bahwa:

- tidak semua masalah butuh label
- clustering bisa memberi insight segmentasi
- scaling penting untuk K-Means
- hasil cluster bukan “jawaban mutlak”, tetapi titik awal eksplorasi

18.6 Template kode umum: pilih model, `fit`, prediksi, evaluasi

Ini bagian yang sangat penting untuk pemula. Di banyak referensi lampiran, pola penggunaan scikit-learn sangat konsisten:

- pilih `X` dan `y`
- split train-test
- buat model
- `fit()`
- `predict()`
- evaluasi sesuai jenis tugas

Template umum untuk supervised learning

Simpan sebagai:

```
src/bab18_template_supervised.py

import pandas as pd
from sklearn.model_selection import train_test_split

# Ganti model sesuai kebutuhan
from sklearn.linear_model import LinearRegression
# from sklearn.linear_model import LogisticRegression
# from sklearn.ensemble import RandomForestClassifier

# 1. Load data
df = pd.read_csv("data/raw/dataset_anda.csv")

# 2. Pilih fitur dan target
X = df[["fitur1", "fitur2"]]
y = df["target"]

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

# 4. Pilih model
model = LinearRegression()

# 5. Training
model.fit(X_train, y_train)

# 6. Prediksi
y_pred = model.predict(X_test)

# 7. Evaluasi
print(y_pred)
```

Template umum untuk clustering

Simpan sebagai:

```
src/bab18_template_clustering.py

import pandas as pd
from sklearn.cluster import KMeans

# 1. Load data
df = pd.read_csv("data/raw/dataset_clustering.csv")

# 2. Pilih fitur
X = df[["fitur1", "fitur2"]]

# 3. Pilih model
model = KMeans(n_clusters=3, random_state=42, n_init=10)

# 4. Training + prediksi cluster
df["cluster"] = model.fit_predict(X)

# 5. Lihat hasil
print(df)
```

Cara memakai template ini

Pembaca cukup mengganti:

- nama file
- nama fitur
- nama target
- jenis model
- metrik evaluasi

Dengan pola seperti ini, pembaca mulai membangun kebiasaan kerja yang konsisten.

18.7 Cara memilih model dari empat contoh ini

Empat kasus di bab ini bisa diringkas menjadi pola pikir yang sangat sederhana:

- **Prediksi harga rumah**
  Targetnya angka → pakai `LinearRegression`
- **Lulus/tidak lulus**
  Targetnya kelas biner → pakai `LogisticRegression`
- **Klasifikasi pelanggan**
  Targetnya kelas, datanya tabular dan agak lebih kompleks → pakai `RandomForestClassifier`
- **Segmentasi pelanggan**
  Tidak ada label, ingin pengelompokan → pakai `KMeans`

Pola ini sangat selaras dengan materi lampiran tentang **supervised vs unsupervised**, **regression vs classification**, *choosing the right algorithm*, dan *model selection* berdasarkan bentuk masalah dan kebutuhan praktis.

18.8 Kesalahan umum pemula saat memilih algoritma

Ada beberapa jebakan yang sering terjadi.

- **Memilih algoritma karena terdengar canggih**
  Padahal masalahnya sederhana dan model sederhana sudah cukup.
- **Tidak membedakan target angka vs target kelas**
  Ini kesalahan paling dasar, tetapi sangat sering terjadi.
- **Memakai KMeans untuk data yang sebenarnya punya label**
  Padahal itu masalah supervised, bukan clustering.
- **Langsung memakai model kompleks tanpa baseline**
  Ini membuat hasil sulit dievaluasi secara jujur.
- **Menganggap model paling akurat pasti terbaik**
  Padahal kadang interpretabilitas lebih penting.
- **Tidak menyesuaikan preprocessing dengan algoritma**
  Misalnya KNN dan KMeans sering lebih butuh scaling dibanding Random Forest.

Referensi lampiran juga menegaskan bahwa pemilihan model harus melihat **performance metrics**, **interpretability**, **computational efficiency**, dan kesesuaian terhadap tujuan analisis.

18.9 Refleksi: memilih algoritma adalah bagian dari berpikir analitis

Bab ini seharusnya membuat pembaca lebih tenang. Memilih algoritma bukan ritual rahasia. Ia adalah proses bertanya:

- target saya apa?
- saya punya label atau tidak?
- saya butuh model yang bisa dijelaskan atau cukup yang kuat?
- data saya sederhana atau cukup kompleks?
- saya sedang membuat baseline, klasifikasi tabular, atau segmentasi?

Dengan pertanyaan seperti itu, pilihan algoritma menjadi jauh lebih rasional. Referensi lampiran juga sangat konsisten menunjukkan bahwa pemilihan model harus selalu dikaitkan dengan **jenis masalah, karakter data, dan tujuan penggunaan**, bukan sekadar preferensi pribadi.

Penutup Bab

Bab ini memberi contoh konkret pemilihan algoritma beserta **source code Python** untuk empat kasus yang sangat umum: **prediksi harga rumah dengan `LinearRegression`**, **lulus/tidak lulus dengan `LogisticRegression`**, **klasifikasi pelanggan dengan `RandomForestClassifier`**, dan **segmentasi pelanggan dengan `KMeans`**. Selain itu, bab ini juga memberi **template kode umum** untuk alur memilih model, `fit`, prediksi, dan evaluasi. Kebutuhan akan bab seperti ini memang sangat relevan dengan materi lampiran tentang **choosing the right algorithm**, **model selection**, **complexity vs interpretability**, dan **supervised vs unsupervised learning**.

Pesan terpenting dari bab ini: algoritma yang cocok adalah algoritma yang **sesuai dengan bentuk masalah, tujuan, dan konteks data Anda**, bukan yang paling sering dibicarakan.

Tantangan kecil untuk pembaca: buat satu file baru yang menjalankan dua model klasifikasi pada dataset pelanggan yang sama — **Logistic Regression** dan **Random Forest** — lalu bandingkan mana yang lebih mudah dijelaskan dan mana yang memberi hasil evaluasi lebih baik.

Bab 19. Supervised Learning I — Regresi

Bab ini memperdalam jalur **prediksi nilai angka** dalam *supervised learning*. Di outline naskah utama, bagian ini memang difokuskan pada **Linear Regression**, **Ridge**, **Lasso**, evaluasi **MAE, MSE, RMSE, R²**, serta pembacaan koefisien model untuk kasus seperti prediksi harga rumah atau penjualan. Pada pengembangan outline yang lebih rinci, **Polynomial Regression** juga sangat relevan sebagai perluasan ringan ketika hubungan data tidak lagi sepenuhnya linear. Ini selaras dengan materi lampiran yang banyak membahas regresi linear, regularisasi, dan evaluasi model dengan scikit-learn.

Secara praktis, bab ini penting karena banyak masalah nyata dalam AI/ML pemula justru berawal dari pertanyaan sederhana seperti:

- berapa harga rumah yang layak diprediksi?
- berapa penjualan bulan depan?
- berapa konsumsi energi?
- berapa nilai mahasiswa berdasarkan kebiasaan belajar?

Semua pertanyaan itu menghasilkan **target numerik**, sehingga keluarga model yang masuk akal adalah **regresi**. Referensi pendukung juga menjelaskan bahwa regresi adalah salah satu fondasi paling penting dalam machine learning karena memberi model yang **mudah dijelaskan**, **mudah diuji**, dan **sangat cocok untuk membangun intuisi prediksi**.

Bab ini dirancang agar pembaca mampu:

- memahami **Linear Regression**
- memahami **Ridge Regression**
- memahami **Lasso Regression**
- mengenal **Polynomial Regression** secara ringan
- tahu **kapan memakai masing-masing**
- menjalankan **contoh source code Python lengkap**

19.1 Apa itu regresi dalam *supervised learning*

Dalam *supervised learning*, regresi dipakai ketika target yang ingin diprediksi berupa **angka kontinu**. Referensi Jain menjelaskan hal ini secara langsung dengan contoh seperti prediksi harga rumah, sedangkan referensi Van Der Post menunjukkan implementasi *Linear Regression* untuk memprediksi harga rumah berdasarkan fitur seperti luas rumah dan jumlah kamar.

Ciri masalah regresi

- target berupa angka
- model belajar dari pasangan fitur–target
- hasil prediksi juga berupa angka

Contoh masalah regresi

- harga rumah
- omzet penjualan
- suhu
- nilai mahasiswa
- biaya operasional

Alur umum regresi

1. baca dataset
2. pilih fitur `X`
3. pilih target `y`
4. split train-test
5. latih model
6. prediksi
7. evaluasi
8. interpretasi

Alur ini sangat konsisten dengan contoh-contoh regresi di lampiran, terutama yang memakai `train_test_split`, `LinearRegression`, `Ridge`, `Lasso`, dan metrik evaluasi regresi.

19.2 Dataset contoh untuk bab ini

Agar pembahasan lebih membumi, kita akan memakai dataset sederhana bertema **harga rumah**.

Simpan sebagai:

```
data/raw/regresi_rumah_lengkap.csv
```

Isi file:

```
luas_rumah,jumlah_kamar,usia_rumah,harga
80,2,20,250000
90,2,18,270000
100,3,15,320000
110,3,12,340000
120,3,10,360000
130,4,9,400000
140,4,8,430000
150,4,7,460000
160,5,6,500000
170,5,5,530000
180,5,4,560000
190,6,3,600000
200,6,2,640000
210,6,2,670000
220,7,1,710000
230,7,1,740000
```

```
240,7,1,780000
250,8,1,820000
```

Dataset ini cukup baik untuk:

- **Linear Regression**
- **Ridge Regression**
- **Lasso Regression**
- **Polynomial Regression** ringan

19.3 Linear Regression

Intuisi

Linear Regression mengasumsikan bahwa target dapat diperkirakan sebagai kombinasi linear dari fitur. Referensi Van Der Post menuliskan bentuk umumnya sebagai:

$$y = \beta 0 + \beta 1 x1 + \beta 2 x2 + ... + \beta n\ xn$$

dengan `β0` sebagai intercept dan `β1 ... βn` sebagai koefisien fitur. Referensi tersebut juga menekankan bahwa model ini sangat baik untuk membangun prediksi yang mudah dijelaskan.

Kapan memakai Linear Regression

Gunakan **Linear Regression** jika:

- Anda ingin baseline regresi yang paling jelas
- hubungan fitur-target tampak cukup linear
- interpretasi koefisien penting
- ingin model yang cepat dan mudah dijelaskan

Source code Python lengkap

Simpan sebagai:

src/bab19_linear_regression.py

```
import pandas as pd
import math
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LinearRegression
from sklearn.metrics import mean_absolute_error, mean_squared_error,
r2_score

# 1. Load data
df = pd.read_csv("data/raw/regresi_rumah_lengkap.csv")

# 2. Pilih fitur dan target
X = df[["luas_rumah", "jumlah_kamar", "usia_rumah"]]
y = df["harga"]

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

# 4. Bangun model
model = LinearRegression()
model.fit(X_train, y_train)

# 5. Prediksi
y_pred = model.predict(X_test)

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

print("=== LINEAR REGRESSION ===")
print("Intercept:", model.intercept_)
print("Koefisien:")
for fitur, coef in zip(X.columns, model.coef_):
    print(f"{fitur}: {coef}")

print("\nEvaluasi:")
print("MAE  :", mae)
print("MSE  :", mse)
print("RMSE :", rmse)
print("R²   :", r2)

hasil = pd.DataFrame({
    "aktual": y_test.values,
    "prediksi": y_pred
})

print("\nHasil prediksi:")
print(hasil)
```

Apa yang dipelajari

Dari model ini, pembaca bisa melihat:

- **intercept** = titik awal prediksi saat semua fitur nol
- **koefisien** = arah dan besar pengaruh fitur
- **MAE, MSE, RMSE, R^2** = kualitas model

Referensi lampiran juga menekankan bahwa *Linear Regression* biasanya dievaluasi dengan **MSE** dan **R^2**, serta bahwa koefisien dan intercept adalah bagian penting dari interpretasi model.

19.4 Ridge Regression

Intuisi

Ridge Regression adalah variasi dari regresi linear yang menambahkan penalti pada besar koefisien. Tujuannya adalah **menstabilkan model** dan mengurangi risiko ketika fitur saling berkorelasi atau model terlalu sensitif. Referensi Knox menjelaskan bahwa Ridge mengganti solusi *least squares* biasa dengan bentuk yang menambahkan matriks identitas berskala λ, dan menyebut bahwa praktik umum adalah melakukan transformasi atau penyetaraan skala fitur sebelum fitting. Referensi Van Der Post juga menjelaskan Ridge sebagai solusi terhadap *multicollinearity* melalui regularisasi.

Kapan memakai Ridge Regression

Gunakan **Ridge Regression** jika:

- ada banyak fitur yang saling berkorelasi
- koefisien model linear tampak tidak stabil
- Anda ingin tetap mempertahankan semua fitur, tetapi dengan penalti
- ingin mengurangi varians model

Catatan penting

Ridge sering lebih masuk akal jika fitur sudah berada pada skala yang sebanding. Itu juga ditegaskan di referensi Knox.

Source code Python lengkap

Simpan sebagai:

src/bab19_ridge_regression.py

```
import pandas as pd
import math
from sklearn.model_selection import train_test_split
from sklearn.linear_model import Ridge
from sklearn.preprocessing import StandardScaler
from sklearn.metrics import mean_absolute_error, mean_squared_error,
r2_score

# 1. Load data
df = pd.read_csv("data/raw/regresi_rumah_lengkap.csv")

# 2. Fitur dan target
X = df[["luas_rumah", "jumlah_kamar", "usia_rumah"]]
y = df["harga"]

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

# 4. Scaling
scaler = StandardScaler()
X_train_scaled = scaler.fit_transform(X_train)
X_test_scaled = scaler.transform(X_test)

# 5. Bangun model Ridge
model = Ridge(alpha=1.0)
model.fit(X_train_scaled, y_train)

# 6. Prediksi
y_pred = model.predict(X_test_scaled)

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

print("=== RIDGE REGRESSION ===")
print("Intercept:", model.intercept_)
print("Koefisien:")
for fitur, coef in zip(X.columns, model.coef_):
    print(f"{fitur}: {coef}")

print("\nEvaluasi:")
print("MAE  :", mae)
print("MSE  :", mse)
print("RMSE :", rmse)
print("R²   :", r2)
```

Apa yang dipelajari

Ridge membantu pembaca melihat bahwa:

- model linear bisa diperhalus dengan regularisasi
- semua fitur tetap dipakai
- koefisien cenderung mengecil, tetapi tidak menjadi nol

Referensi Lim juga menegaskan bahwa Ridge termasuk model yang umum di tuning dengan validasi untuk mencari R^2 terbaik.

19.5 Lasso Regression

Intuisi

Lasso Regression juga menambahkan penalti, tetapi berbeda dengan Ridge karena ia memakai penalti absolut. Referensi Knox menjelaskan bahwa Lasso meminimalkan RSS ditambah penalti jumlah absolut koefisien, dan memiliki sifat penting: **melakukan *automatic feature selection***, karena beberapa koefisien bisa menjadi nol.

Kapan memakai Lasso Regression

Gunakan **Lasso Regression** jika:

- Anda ingin model yang lebih ringkas
- Anda curiga tidak semua fitur benar-benar penting
- Anda ingin regularisasi sekaligus seleksi fitur
- ingin melihat fitur mana yang mungkin bisa “dimatikan” model

Source code Python lengkap

Simpan sebagai:

src/bab19_lasso_regression.py

```
import pandas as pd
import math
from sklearn.model_selection import train_test_split
from sklearn.linear_model import Lasso
from sklearn.preprocessing import StandardScaler
from sklearn.metrics import mean_absolute_error, mean_squared_error,
r2_score

# 1. Load data
df = pd.read_csv("data/raw/regresi_rumah_lengkap.csv")

# 2. Fitur dan target
X = df[["luas_rumah", "jumlah_kamar", "usia_rumah"]]
y = df["harga"]

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

# 4. Scaling
scaler = StandardScaler()
X_train_scaled = scaler.fit_transform(X_train)
X_test_scaled = scaler.transform(X_test)

# 5. Bangun model Lasso
model = Lasso(alpha=0.1)
model.fit(X_train_scaled, y_train)

# 6. Prediksi
y_pred = model.predict(X_test_scaled)

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

print("=== LASSO REGRESSION ===")
print("Intercept:", model.intercept_)
print("Koefisien:")
for fitur, coef in zip(X.columns, model.coef_):
    print(f"{fitur}: {coef}")

print("\nEvaluasi:")
print("MAE  :", mae)
print("MSE  :", mse)
print("RMSE :", rmse)
print("R²   :", r2)
```

Apa yang dipelajari

Lasso membantu pembaca melihat bahwa:

- regularisasi bisa sekaligus menjadi seleksi fitur
- beberapa koefisien bisa sangat kecil atau nol
- model bisa menjadi lebih sederhana

Ini sangat sesuai dengan penjelasan Knox bahwa Lasso punya efek seleksi fitur otomatis.

19.6 Polynomial Regression secara ringan

Mengapa perlu Polynomial Regression

Tidak semua hubungan antara fitur dan target bersifat linear lurus. Kadang data membentuk kurva. Dalam kasus seperti itu, **Polynomial Regression** bisa menjadi perluasan ringan dari regresi linear. Referensi Van Der Post menjelaskan bahwa *Polynomial Regression* menambahkan suku pangkat ke dalam fitur agar model linear dapat memetakan pola nonlinier, tetapi juga mengingatkan bahwa derajat polinomial yang terlalu tinggi meningkatkan risiko *overfitting*.

Kapan memakai Polynomial Regression

Gunakan **Polynomial Regression** jika:

- scatter plot menunjukkan pola melengkung
- model linear biasa tampak kurang pas
- Anda ingin model sedikit lebih fleksibel, tetapi masih dalam keluarga linear terhadap fitur hasil transformasi

Catatan penting

Polynomial Regression tetap menggunakan **LinearRegression** di tahap fitting, tetapi fitur lebih dulu diubah dengan **PolynomialFeatures**. Ini juga ditunjukkan jelas di referensi lampiran.

Dataset contoh nonlinier ringan

Simpan sebagai:

```
data/raw/regresi_polynomial_ringan.csv
```

Isi file:

```
x,y
1,8
2,11
```

```
3,17
4,26
5,38
6,53
7,71
8,92
9,116
10,143
```

Source code Python lengkap

Simpan sebagai:

```
src/bab19_polynomial_regression.py

import pandas as pd
import math
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import PolynomialFeatures, StandardScaler
from sklearn.linear_model import LinearRegression
from sklearn.metrics import mean_absolute_error, mean_squared_error,
r2_score

# 1. Load data
df = pd.read_csv("data/raw/regresi_polynomial_ringan.csv")

X = df[["x"]]
y = df["y"]

# 2. Buat fitur polynomial degree 2
poly = PolynomialFeatures(degree=2, include_bias=False)
X_poly = poly.fit_transform(X)

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

# 4. Scaling opsional
scaler = StandardScaler()
X_train_scaled = scaler.fit_transform(X_train)
X_test_scaled = scaler.transform(X_test)

# 5. Model
model = LinearRegression()
model.fit(X_train_scaled, y_train)

# 6. Prediksi
y_pred = model.predict(X_test_scaled)

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

print("=== POLYNOMIAL REGRESSION (DEGREE 2) ===")
```

```
print("Fitur hasil transformasi:", poly.get_feature_names_out(["x"]))
print("Intercept:", model.intercept_)
print("Koefisien:", model.coef_)

print("\nEvaluasi:")
print("MAE  :", mae)
print("MSE  :", mse)
print("RMSE :", rmse)
print("R²   :", r2)
```

Apa yang dipelajari

Dari contoh ini, pembaca bisa memahami bahwa:

- pola nonlinier ringan dapat ditangani dengan transformasi polinomial
- model menjadi lebih fleksibel
- tetapi kompleksitas juga naik
- risiko *overfitting* harus dijaga

Itu sangat selaras dengan penjelasan referensi Van Der Post.

19.7 Kapan memakai masing-masing

Ini bagian yang paling praktis.

1. Linear Regression

Pakai jika:

- ingin baseline regresi yang mudah dipahami
- hubungan tampak cukup linear
- koefisien ingin dibaca
- masalah masih sederhana

Referensi lampiran berulang kali menempatkan Linear Regression sebagai titik mulai utama untuk prediksi numerik.

2. Ridge Regression

Pakai jika:

- ada indikasi *multicollinearity*
- fitur saling berkorelasi
- ingin model lebih stabil
- tetap ingin mempertahankan semua fitur

Referensi Knox dan Van Der Post menjelaskan Ridge sebagai solusi regularisasi yang menstabilkan koefisien.

3. Lasso Regression

Pakai jika:

- ingin regularisasi sekaligus seleksi fitur
- curiga ada fitur yang kurang penting
- ingin model lebih ringkas

Referensi Knox menekankan efek seleksi fitur otomatis pada Lasso.

4. Polynomial Regression

Pakai jika:

- hubungan tampak melengkung
- model linear terlalu sederhana
- ingin memperkaya fleksibilitas model tanpa langsung pindah ke model yang jauh lebih kompleks

Referensi Van Der Post juga menekankan bahwa Polynomial Regression berguna untuk menangkap pola non linier, tetapi harus hati-hati terhadap *overfitting*.

19.8 Perbandingan intuitif

Secara sangat sederhana:

- **Linear Regression**
  paling mudah dijelaskan, paling sederhana
- **Ridge Regression**
  seperti Linear Regression yang “lebih disiplin” karena koefisien dikecilkan
- **Lasso Regression**
  seperti Linear Regression yang bisa “mematikan” fitur tertentu
- **Polynomial Regression**
  seperti Linear Regression yang diberi fitur lengkung tambahan agar lebih fleksibel

Ini cara berpikir yang cukup baik untuk pemula sebelum masuk terlalu jauh ke matematika regulasi dan optimalisasi.

19.9 Source code Python lengkap: perbandingan empat model

Simpan sebagai:

```
src/bab19_perbandingan_regresi.py
```

```
import pandas as pd
import math
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler, PolynomialFeatures
from sklearn.linear_model import LinearRegression, Ridge, Lasso
from sklearn.metrics import mean_squared_error, r2_score

# ==============================
# A. Dataset linear
# ==============================
df = pd.read_csv("data/raw/regresi_rumah_lengkap.csv")

X = df[["luas_rumah", "jumlah_kamar", "usia_rumah"]]
y = df["harga"]

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

scaler = StandardScaler()
X_train_scaled = scaler.fit_transform(X_train)
X_test_scaled = scaler.transform(X_test)

models = {
    "LinearRegression": LinearRegression(),
    "Ridge": Ridge(alpha=1.0),
    "Lasso": Lasso(alpha=0.1)
}

print("=== PERBANDINGAN MODEL REGRESI LINEAR / RIDGE / LASSO ===")
for nama, model in models.items():
    if nama == "LinearRegression":
        model.fit(X_train, y_train)
        y_pred = model.predict(X_test)
    else:
        model.fit(X_train_scaled, y_train)
        y_pred = model.predict(X_test_scaled)

    rmse = math.sqrt(mean_squared_error(y_test, y_pred))
    r2 = r2_score(y_test, y_pred)

    print(f"\nModel: {nama}")
    print("RMSE:", rmse)
    print("R²  :", r2)

# ==============================
# B. Dataset polynomial
```

```
# ==============================
df_poly = pd.read_csv("data/raw/regresi_polynomial_ringan.csv")

Xp = df_poly[["x"]]
yp = df_poly["y"]

poly = PolynomialFeatures(degree=2, include_bias=False)
Xp_poly = poly.fit_transform(Xp)

Xp_train, Xp_test, yp_train, yp_test = train_test_split(
    Xp_poly, yp, test_size=0.2, random_state=42
)

scaler_poly = StandardScaler()
Xp_train_scaled = scaler_poly.fit_transform(Xp_train)
Xp_test_scaled = scaler_poly.transform(Xp_test)

poly_model = LinearRegression()
poly_model.fit(Xp_train_scaled, yp_train)
yp_pred = poly_model.predict(Xp_test_scaled)

rmse_poly = math.sqrt(mean_squared_error(yp_test, yp_pred))
r2_poly = r2_score(yp_test, yp_pred)

print("\n=== POLYNOMIAL REGRESSION ===")
print("Fitur transformasi:", poly.get_feature_names_out(["x"]))
print("RMSE:", rmse_poly)
print("R²  :", r2_poly)
```

Program ini membantu pembaca membandingkan empat pendekatan dalam satu tempat.

19.10 Kesalahan umum pemula

Ada beberapa jebakan yang sering muncul.

- **Memakai Polynomial Regression terlalu tinggi derajatnya**
  sehingga model justru *overfit*.
- **Memakai Ridge/Lasso tanpa memperhatikan scaling**
  padahal regularisasi sangat sensitif pada skala fitur. Ini juga disinggung oleh referensi Knox dan contoh pipeline Ridge di referensi lain.
- **Mengira Ridge dan Lasso selalu lebih baik**
  padahal tergantung konteks data.
- **Tidak membandingkan dengan baseline Linear Regression**
  sehingga sulit tahu apakah regularisasi benar-benar membantu.
- **Terlalu fokus pada skor, lupa interpretasi**
  padahal salah satu kekuatan regresi justru pada interpretabilitas.

19.11 Refleksi: regresi adalah laboratorium yang sangat baik untuk belajar model

Bab ini seharusnya membuat pembaca melihat bahwa regresi bukan sekadar satu algoritma, tetapi satu keluarga model dengan karakter berbeda:

- ada yang sangat sederhana dan mudah dijelaskan
- ada yang lebih stabil melalui regularisasi
- ada yang sekaligus menyeleksi fitur
- ada yang lebih fleksibel untuk pola nonlinier

Referensi lampiran juga sangat konsisten menempatkan regresi sebagai fondasi machine learning yang kuat, karena dari sinilah pembaca belajar banyak konsep penting sekaligus: **koefisien, regularisasi, evaluasi, interpretasi, multicollinearity, dan *bias-variance tradeoff***.

Penutup Bab

Bab ini memperkenalkan **Supervised Learning I — Regresi** melalui empat pendekatan utama: **Linear Regression**, **Ridge Regression**, **Lasso Regression**, dan **Polynomial Regression** secara ringan. Pembaca juga belajar **kapan memakai masing-masing** serta menjalankan **contoh source code Python lengkap** dengan scikit-learn. Seluruh pembahasan ini sangat relevan dengan materi lampiran yang membahas regresi linear, regularisasi, evaluasi model, dan penggunaan NumPy, Pandas, serta scikit-learn dalam workflow prediksi numerik.

Pesan terpenting dari bab ini: mulai dari model regresi yang paling sederhana dan paling bisa Anda pahami, lalu naikkan kompleksitas hanya jika data benar-benar menuntutnya.

Tantangan kecil untuk pembaca: modifikasi file `bab19_perbandingan_regresi.py` dengan menambahkan satu fitur baru buatan, misalnya `luas_per_kamar = luas_rumah / jumlah_kamar`, lalu bandingkan apakah metrik **RMSE** dan **R^2** berubah pada Linear, Ridge, dan Lasso.

Bab 20. Evaluasi Model Regresi

Bab ini sangat penting karena pada tahap inilah pembaca belajar satu hal yang sering dilupakan pemula: **model regresi yang bisa berjalan belum tentu model yang baik**. Di outline naskah utama, Bab 20 memang difokuskan pada **MAE, MSE, RMSE, R², visualisasi prediksi vs aktual, error analysis dasar, dan cross-validation sederhana**. Susunan ini sangat selaras dengan materi lampiran yang menempatkan evaluasi model regresi sebagai tahap wajib setelah training, bukan pelengkap opsional. Referensi lampiran juga konsisten menekankan bahwa evaluasi membantu kita melihat **seberapa dekat prediksi model dengan nilai aktual**, seberapa stabil model pada data baru, dan apakah model masih layak diperbaiki lebih lanjut.

Materi ini juga didukung kuat oleh referensi lain di lampiran. Buku *Machine Learning in Business Finance Using Python* menjelaskan bahwa **R²** dan **RMSE** adalah ukuran penting untuk menilai performa prediksi, terutama pada data validasi dan test, serta menegaskan bahwa skor train yang tinggi saja tidak cukup karena model bisa mengalami **overfitting** bila performa di validation/test menurun. Buku *Python for Data Analysis* juga menempatkan **cross-validation** sebagai langkah penting untuk menilai konsistensi model di beberapa lipatan data, bukan hanya pada satu kali split train-test.

Bab ini dirancang agar pembaca mampu:

- memahami **MAE**
- memahami **MSE**
- memahami **RMSE**
- memahami **R²**
- membuat **visualisasi prediksi vs aktual**
- melakukan **error analysis dasar**
- menjalankan **cross-validation sederhana**
- membaca hasil evaluasi dengan lebih matang sebelum menyimpulkan model sudah “bagus”

20.1 Mengapa evaluasi model regresi itu penting

Dalam regresi, model mencoba memprediksi **angka**. Karena itu, kita perlu tahu bukan hanya apakah arah prediksi sudah benar, tetapi juga **seberapa besar selisih antara nilai aktual dan nilai prediksi**. Referensi lampiran menekankan bahwa evaluasi regresi harus memakai beberapa metrik sekaligus, karena tiap metrik memberi sudut pandang berbeda terhadap error model. MAE memberi gambaran error rata-rata yang mudah dipahami, MSE lebih sensitif terhadap error besar, RMSE mengembalikan error ke satuan asli target, dan R² membantu melihat seberapa besar variasi target dijelaskan model.

Kalau evaluasi diabaikan, kita tidak tahu:

- apakah model benar-benar berguna
- apakah error-nya kecil atau besar
- apakah model cukup stabil
- apakah model lebih baik dari baseline

- apakah model perlu diiterasi lagi

Itulah sebabnya evaluasi model bukan langkah tambahan. Evaluasi adalah **jantung penilaian kualitas model**.

20.2 Dataset contoh untuk bab ini

Agar pembahasan lebih hidup, kita akan memakai dataset sederhana bertema prediksi harga rumah.

Simpan sebagai:

```
data/raw/evaluasi_regresi_rumah.csv
```

Isi file:

```
luas_rumah,jumlah_kamar,usia_rumah,harga
80,2,20,250000
90,2,18,270000
100,3,15,320000
110,3,12,340000
120,3,10,360000
130,4,9,400000
140,4,8,430000
150,4,7,460000
160,5,6,500000
170,5,5,530000
180,5,4,560000
190,6,3,600000
200,6,2,640000
210,6,2,670000
220,7,1,710000
230,7,1,740000
240,7,1,780000
250,8,1,820000
```

Dataset ini cukup sederhana untuk:

- melatih model regresi awal
- menghitung metrik evaluasi
- membuat visualisasi prediksi vs aktual
- melakukan analisis residual
- mencoba cross-validation ringan

20.3 Menyiapkan model awal untuk dievaluasi

Sebelum menghitung metrik, tentu kita perlu melatih model terlebih dahulu. Untuk bab ini, kita akan memakai **Linear Regression** sebagai model dasar. Referensi lampiran memberi contoh yang sangat serupa: memuat dataset, memisahkan `X` dan `y`, melakukan `train_test_split`, melatih `LinearRegression`, lalu menghitung metrik evaluasi seperti MSE dan R^2.

Simpan sebagai:

```
src/bab20_model_awal_regresi.py

import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LinearRegression

# 1. Load data
df = pd.read_csv("data/raw/evaluasi_regresi_rumah.csv")

# 2. Fitur dan target
X = df[["luas_rumah", "jumlah_kamar", "usia_rumah"]]
y = df["harga"]

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

# 4. Latih model
model = LinearRegression()
model.fit(X_train, y_train)

# 5. Prediksi
y_pred = model.predict(X_test)

print("Intercept:", model.intercept_)
print("Koefisien:")
for fitur, coef in zip(X.columns, model.coef_):
    print(f"{fitur}: {coef}")

hasil = pd.DataFrame({
    "aktual": y_test.values,
    "prediksi": y_pred
})

print("\nHasil prediksi:")
print(hasil)
```

Program ini menjadi fondasi untuk seluruh evaluasi berikutnya.

20.4 MAE

Apa itu MAE

MAE (*Mean Absolute Error*) adalah rata-rata dari nilai absolut selisih antara prediksi dan nilai aktual. Referensi lampiran menekankan bahwa MAE adalah metrik yang sangat mudah dipahami karena langsung menunjukkan rata-rata besarnya kesalahan prediksi tanpa memperhatikan arah error.

Secara intuitif, MAE menjawab pertanyaan:

> **rata-rata model saya meleset berapa satuan dari nilai sebenarnya?**

Kalau targetnya harga rumah dalam rupiah, maka MAE juga akan berada dalam satuan rupiah.

Kelebihan MAE

- mudah dijelaskan
- tidak melebih-lebihkan error besar
- cocok sebagai ukuran praktis untuk “rata-rata meleset”

Source code Python lengkap

Simpan sebagai:

```
src/bab20_mae.py

import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LinearRegression
from sklearn.metrics import mean_absolute_error

df = pd.read_csv("data/raw/evaluasi_regresi_rumah.csv")

X = df[["luas_rumah", "jumlah_kamar", "usia_rumah"]]
y = df["harga"]

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

model = LinearRegression()
model.fit(X_train, y_train)

y_pred = model.predict(X_test)

mae = mean_absolute_error(y_test, y_pred)

print("MAE:", mae)
```

Cara membaca MAE

Kalau MAE = 12.000, artinya:

> **rata-rata prediksi harga rumah meleset sekitar 12.000 rupiah dari harga aktual**

Tentu interpretasi “baik” atau “buruk” tetap harus melihat konteks harga rumahnya.

20.5 MSE

Apa itu MSE

MSE (*Mean Squared Error*) adalah rata-rata error kuadrat. Referensi lampiran menjelaskan bahwa MSE memberi penalti lebih besar pada error yang besar, sehingga sangat berguna bila kita ingin model “lebih takut” terhadap kesalahan ekstrem.

Secara intuitif, MSE menjawab:

> **seberapa besar error model saya jika kesalahan besar diberi hukuman lebih berat?**

Karakter MSE

- sensitif pada outlier
- cocok ketika error besar benar-benar tidak diinginkan
- satuannya menjadi kuadrat dari satuan asli target, sehingga kurang intuitif untuk dibaca langsung

Source code Python lengkap

Simpan sebagai:

src/bab20_mse.py

```
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LinearRegression
from sklearn.metrics import mean_squared_error

df = pd.read_csv("data/raw/evaluasi_regresi_rumah.csv")

X = df[["luas_rumah", "jumlah_kamar", "usia_rumah"]]
y = df["harga"]

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

model = LinearRegression()
model.fit(X_train, y_train)

y_pred = model.predict(X_test)

mse = mean_squared_error(y_test, y_pred)

print("MSE:", mse)
```

Cara membaca MSE

MSE lebih berguna untuk **membandingkan model**, bukan untuk penjelasan awam. Kalau dua model diuji pada target yang sama, model dengan MSE lebih kecil biasanya lebih baik dalam arti error kuadratnya lebih rendah.

20.6 RMSE

Apa itu RMSE

RMSE (*Root Mean Squared Error*) adalah akar dari MSE. Referensi lampiran menjelaskan bahwa RMSE mengembalikan error ke satuan asli target, sehingga lebih mudah dibaca daripada MSE, tetapi tetap mempertahankan sensitivitas terhadap error besar.

Secara intuitif, RMSE menjawab:

> **kalau saya ingin error model dalam satuan asli target, tetapi tetap ingin memberi penalti lebih kuat pada error besar, berapa nilainya?**

Kelebihan RMSE

- satuan sama dengan target
- tetap sensitif terhadap error besar
- sangat sering dipakai dalam evaluasi regresi

Source code Python lengkap

Simpan sebagai:

```
src/bab20_rmse.py
```

```
import pandas as pd
import math
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LinearRegression
from sklearn.metrics import mean_squared_error

df = pd.read_csv("data/raw/evaluasi_regresi_rumah.csv")

X = df[["luas_rumah", "jumlah_kamar", "usia_rumah"]]
y = df["harga"]

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

model = LinearRegression()
model.fit(X_train, y_train)

y_pred = model.predict(X_test)

mse = mean_squared_error(y_test, y_pred)
rmse = math.sqrt(mse)

print("RMSE:", rmse)
```

Cara membaca RMSE

Kalau RMSE = 15.000, artinya model punya error prediksi tipikal sekitar 15.000 dalam satuan harga rumah, dengan error besar diberi bobot lebih kuat daripada MAE.

20.7 R^2

Apa itu R^2

R^2 (*coefficient of determination*) mengukur proporsi variasi target yang dapat dijelaskan oleh model. Referensi lampiran menegaskan bahwa R^2 membantu melihat **daya jelaskan model**, dan nilai yang lebih tinggi menunjukkan model menjelaskan variasi target dengan lebih baik.

Intuisi sederhana

Kalau $R^2 = 0.85$, artinya kira-kira:

> **sekitar 85% variasi harga rumah dalam data ini bisa dijelaskan oleh fitur yang dipakai model**

Catatan penting

R^2 bukan segalanya. Nilai R^2 tinggi tidak otomatis berarti model sempurna. Referensi lampiran juga menekankan bahwa R^2 harus dibaca bersama metrik error dan performa pada data validation/test, bukan hanya pada data train.

Source code Python lengkap

Simpan sebagai:

```
src/bab20_r2.py
```

```
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LinearRegression
from sklearn.metrics import r2_score

df = pd.read_csv("data/raw/evaluasi_regresi_rumah.csv")

X = df[["luas_rumah", "jumlah_kamar", "usia_rumah"]]
y = df["harga"]

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

model = LinearRegression()
model.fit(X_train, y_train)

y_pred = model.predict(X_test)
r2 = r2_score(y_test, y_pred)
print("R²:", r2)
```

Cara membaca R^2

- mendekati **1** → model menjelaskan variasi target dengan baik
- mendekati **0** → model lemah menjelaskan variasi
- bisa negatif pada kondisi tertentu bila model sangat buruk dibanding baseline

20.8 Menghitung semua metrik sekaligus

Dalam praktik nyata, kita hampir selalu menghitung beberapa metrik sekaligus. Itu juga yang ditunjukkan konsisten oleh materi lampiran.

Simpan sebagai:

```
src/bab20_metrik_regresi_lengkap.py

import pandas as pd
import math
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LinearRegression
from sklearn.metrics import mean_absolute_error, mean_squared_error,
r2_score

df = pd.read_csv("data/raw/evaluasi_regresi_rumah.csv")

X = df[["luas_rumah", "jumlah_kamar", "usia_rumah"]]
y = df["harga"]

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

model = LinearRegression()
model.fit(X_train, y_train)

y_pred = model.predict(X_test)

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

print("=== METRIK EVALUASI REGRESI ===")
print("MAE  :", mae)
print("MSE  :", mse)
print("RMSE :", rmse)
print("R²   :", r2)
```

20.9 Visualisasi prediksi vs aktual

Ini salah satu cara paling kuat untuk membuat evaluasi model terasa hidup. Referensi lampiran menekankan bahwa visualisasi dapat membantu kita melihat seberapa dekat prediksi model terhadap nilai aktual, bukan hanya mengandalkan satu angka metrik. Residual analysis juga disebut sebagai komponen penting karena membuka pola dan anomali yang tidak selalu tampak dari metrik ringkas saja.

A. Scatter prediksi vs aktual

Simpan sebagai:

```
src/bab20_visual_prediksi_vs_aktual.py

import pandas as pd
import matplotlib.pyplot as plt
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LinearRegression

df = pd.read_csv("data/raw/evaluasi_regresi_rumah.csv")

X = df[["luas_rumah", "jumlah_kamar", "usia_rumah"]]
y = df["harga"]

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

model = LinearRegression()
model.fit(X_train, y_train)

y_pred = model.predict(X_test)

plt.figure(figsize=(7, 5))
plt.scatter(y_test, y_pred)
plt.xlabel("Nilai Aktual")
plt.ylabel("Nilai Prediksi")
plt.title("Prediksi vs Aktual")
plt.grid(True)
plt.tight_layout()
plt.show()
```

Cara membaca grafik ini

Kalau model bagus, titik-titik akan cenderung berkumpul dekat garis imajiner diagonal:

aktual = prediksi

Semakin jauh titik dari garis ideal itu, semakin besar error prediksi.

B. Versi dengan garis diagonal pembanding

```
import pandas as pd
import matplotlib.pyplot as plt
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LinearRegression

df = pd.read_csv("data/raw/evaluasi_regresi_rumah.csv")

X = df[["luas_rumah", "jumlah_kamar", "usia_rumah"]]
y = df["harga"]

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

model = LinearRegression()
model.fit(X_train, y_train)

y_pred = model.predict(X_test)

plt.figure(figsize=(7, 5))
plt.scatter(y_test, y_pred)
plt.plot([y_test.min(), y_test.max()], [y_test.min(), y_test.max()])
plt.xlabel("Nilai Aktual")
plt.ylabel("Nilai Prediksi")
plt.title("Prediksi vs Aktual dengan Garis Ideal")
plt.grid(True)
plt.tight_layout()
plt.show()
```

Ini biasanya lebih mudah dibaca oleh pemula.

20.10 *Error analysis* dasar

Metrik memberi ringkasan, tetapi *error analysis* membantu kita melihat **di mana** model meleset. Referensi lampiran menegaskan bahwa **residual analysis** adalah komponen penting evaluasi model regresi, karena dapat mengungkap pola yang menunjukkan masalah asumsi, kualitas data, atau bentuk hubungan yang tidak tertangkap model.

A. Menghitung residual

Residual adalah:

aktual - prediksi

Source code Python lengkap

Simpan sebagai:

```
src/bab20_error_analysis_dasar.py
```

```
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LinearRegression

df = pd.read_csv("data/raw/evaluasi_regresi_rumah.csv")

X = df[["luas_rumah", "jumlah_kamar", "usia_rumah"]]
y = df["harga"]

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

model = LinearRegression()
model.fit(X_train, y_train)

y_pred = model.predict(X_test)

hasil = pd.DataFrame({
    "aktual": y_test.values,
    "prediksi": y_pred
})

hasil["residual"] = hasil["aktual"] - hasil["prediksi"]
hasil["abs_error"] = hasil["residual"].abs()

print("=== ERROR ANALYSIS DASAR ===")
print(hasil)

print("\nError terbesar:")
print(hasil.sort_values(by="abs_error", ascending=False).head())
```

B. Plot residual

```
import pandas as pd
import matplotlib.pyplot as plt
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LinearRegression

df = pd.read_csv("data/raw/evaluasi_regresi_rumah.csv")

X = df[["luas_rumah", "jumlah_kamar", "usia_rumah"]]
y = df["harga"]

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

model = LinearRegression()
model.fit(X_train, y_train)

y_pred = model.predict(X_test)

residual = y_test.values - y_pred

plt.figure(figsize=(7, 5))
plt.axhline(y=0)
plt.scatter(y_pred, residual)
plt.xlabel("Nilai Prediksi")
plt.ylabel("Residual")
plt.title("Plot Residual")
plt.grid(True)
plt.tight_layout()
plt.show()
```

Cara membaca residual

Residual yang sehat biasanya:

- tersebar cukup acak di sekitar nol
- tidak membentuk pola melengkung yang jelas
- tidak menunjukkan bahwa error selalu membesar pada rentang tertentu

Kalau residual membentuk pola kuat, itu bisa menandakan bahwa model linear terlalu sederhana atau ada masalah fitur.

20.11 *Cross-validation* sederhana

Satu kali split train-test bisa terlalu bergantung pada kebetulan pembagian data. Karena itu, **cross-validation** sangat berguna. Referensi lampiran menjelaskan bahwa *cross-validation* membagi data ke beberapa fold, lalu model dilatih dan diuji berulang pada subset berbeda. Ini memberi penilaian yang lebih stabil tentang kemampuan *generalization* model.

A. Intuisi *k-fold cross-validation*

Misalnya `cv=5`:

- data dibagi menjadi 5 bagian
- model dilatih 5 kali
- setiap bagian bergantian menjadi validation fold
- hasil akhirnya dirata-ratakan

Referensi Lim juga menekankan bahwa pada k-fold, yang penting bukan hanya mean skor, tetapi juga **dispersion** atau penyebaran skor antar-fold. Jika rata-rata bagus tetapi variasinya tinggi, model belum tentu stabil.

B. Source code Python lengkap

Simpan sebagai:

```
src/bab20_cross_validation_sederhana.py

import pandas as pd
from sklearn.model_selection import cross_val_score
from sklearn.linear_model import LinearRegression

df = pd.read_csv("data/raw/evaluasi_regresi_rumah.csv")

X = df[["luas_rumah", "jumlah_kamar", "usia_rumah"]]
y = df["harga"]

model = LinearRegression()

scores_r2 = cross_val_score(model, X, y, cv=5, scoring="r2")
scores_neg_mse = cross_val_score(model, X, y, cv=5,
scoring="neg_mean_squared_error")

print("=== CROSS-VALIDATION ===")
print("R² per fold:", scores_r2)
print("Mean R²    :", scores_r2.mean())
print("Std R²     :", scores_r2.std())

print("\nNegative MSE per fold:", scores_neg_mse)
print("Mean Negative MSE    :", scores_neg_mse.mean())
```

C. Kenapa neg_mean_squared_error negatif

Di scikit-learn, beberapa metrik error dibuat negatif agar konsisten dengan aturan umum “skor lebih besar lebih baik”. Jadi untuk membaca MSE yang sebenarnya, kita bisa membalik tandanya.

```
mse_cv = -scores_neg_mse.mean()
print("Mean CV MSE:", mse_cv)
```

D. Kapan cross-validation berguna

- dataset tidak terlalu besar
- ingin evaluasi yang lebih stabil
- ingin membandingkan beberapa model
- ingin memeriksa apakah skor model terlalu bergantung pada satu split data

20.12 Source code Python lengkap: evaluasi model regresi terpadu

Berikut satu program lengkap yang menggabungkan semua inti bab ini.

Simpan sebagai:

```
src/bab20_evaluasi_regresi_terpadu.py

import pandas as pd
import math
import matplotlib.pyplot as plt
from sklearn.model_selection import train_test_split, cross_val_score
from sklearn.linear_model import LinearRegression
from sklearn.metrics import mean_absolute_error, mean_squared_error,
r2_score

print("=" * 80)
print("BAB 20 - EVALUASI MODEL REGRESI")
print("=" * 80)

# 1. Load data
df = pd.read_csv("data/raw/evaluasi_regresi_rumah.csv")

print("\n[1] DATA AWAL")
print(df.head())

# 2. Fitur dan target
X = df[["luas_rumah", "jumlah_kamar", "usia_rumah"]]
y = df["harga"]

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

# 4. Latih model
model = LinearRegression()
model.fit(X_train, y_train)

# 5. Prediksi
y_pred = model.predict(X_test)

# 6. Metrik evaluasi
```

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

print("\n[2] METRIK EVALUASI")
print("MAE  :", mae)
print("MSE  :", mse)
print("RMSE :", rmse)
print("R²   :", r2)

# 7. Prediksi vs aktual
hasil = pd.DataFrame({
    "aktual": y_test.values,
    "prediksi": y_pred
})
hasil["residual"] = hasil["aktual"] - hasil["prediksi"]
hasil["abs_error"] = hasil["residual"].abs()

print("\n[3] HASIL PREDIKSI")
print(hasil)

print("\n[4] ERROR TERBESAR")
print(hasil.sort_values(by="abs_error", ascending=False).head())

# 8. Visualisasi prediksi vs aktual
plt.figure(figsize=(7, 5))
plt.scatter(y_test, y_pred)
plt.plot([y_test.min(), y_test.max()], [y_test.min(), y_test.max()])
plt.xlabel("Nilai Aktual")
plt.ylabel("Nilai Prediksi")
plt.title("Prediksi vs Aktual")
plt.grid(True)
plt.tight_layout()
plt.show()

# 9. Visualisasi residual
plt.figure(figsize=(7, 5))
plt.axhline(y=0)
plt.scatter(y_pred, hasil["residual"])
plt.xlabel("Nilai Prediksi")
plt.ylabel("Residual")
plt.title("Plot Residual")
plt.grid(True)
plt.tight_layout()
plt.show()

# 10. Cross-validation
scores_r2 = cross_val_score(model, X, y, cv=5, scoring="r2")
scores_neg_mse = cross_val_score(model, X, y, cv=5,
scoring="neg_mean_squared_error")

print("\n[5] CROSS-VALIDATION")
```

```
print("R² per fold :", scores_r2)
print("Mean R²     :", scores_r2.mean())
print("Std R²      :", scores_r2.std())
print("Mean CV MSE :", -scores_neg_mse.mean())

print("\nSELESAI. Anda sudah mengevaluasi model regresi secara lebih
lengkap.")
print("=" * 80)
```

20.13 Cara membaca hasil evaluasi dengan sehat

Agar pembaca tidak terjebak hanya membaca angka, berikut cara berpikir yang saya sarankan.

- **MAE**
  bagus untuk menjawab: rata-rata model meleset berapa
- **MSE**
  bagus untuk membandingkan model jika error besar harus dihukum lebih keras
- **RMSE**
  bagus untuk membaca error dalam satuan asli target
- **R²**
  bagus untuk melihat daya jelaskan model, tetapi jangan dibaca sendirian
- **Prediksi vs aktual**
  bagus untuk melihat apakah model mendekati garis ideal
- **Residual plot**
  bagus untuk mencari pola error yang tidak sehat
- **Cross-validation**
  bagus untuk melihat apakah model stabil di beberapa split data

Referensi lampiran juga sangat jelas bahwa evaluasi model harus dilihat sebagai **proses multi-sudut**, bukan satu angka tunggal.

20.14 Kesalahan umum pemula

Ada beberapa jebakan yang sangat sering terjadi.

- **Hanya melihat R²**
  Padahal error aktual bisa tetap besar.
- **Tidak melihat satuan error**
  Padahal MAE/RMSE jauh lebih bermakna dalam konteks asli target.
- **Tidak membuat plot residual**
  Padahal pola residual bisa menunjukkan masalah model.
- **Puas dengan satu train-test split**
  Padahal hasil bisa kebetulan bagus atau buruk.
- **Membandingkan metrik di dataset berbeda**
  Padahal itu sering tidak adil.
- **Menganggap skor train tinggi berarti model bagus**
  Padahal yang lebih penting adalah validation/test performance. Ini juga ditegaskan di materi lampiran tentang overfitting dan validation/test R².

20.15 Refleksi: evaluasi model adalah latihan kejujuran analitis

Bab ini seharusnya membuat pembaca melihat bahwa evaluasi model bukan sekadar ritual teknis. Evaluasi adalah **latihan kejujuran**. Saat model diberi data baru, apakah ia tetap bekerja baik? Saat error dilihat satu per satu, apakah ada pola yang merisaukan? Saat skor dirata-ratakan lewat cross-validation, apakah model tetap stabil?

Itulah sebabnya referensi lampiran menekankan evaluasi sebagai dasar **feedback loop** untuk perbaikan model, pemilihan metrik yang sesuai konteks, dan keseimbangan antara akurasi serta kompleksitas model.

Penutup Bab

Bab ini membahas **evaluasi model regresi** secara lebih lengkap melalui **MAE, MSE, RMSE, R^2, visualisasi prediksi vs aktual, error analysis dasar, dan cross-validation sederhana**. Seluruh topik ini sangat relevan dengan materi lampiran yang menjelaskan metrik evaluasi regresi, pentingnya residual analysis, dan peran cross-validation untuk menilai *generalization* model secara lebih stabil.

Pesan terpenting dari bab ini: model regresi yang baik bukan model yang sekadar memberi angka prediksi, tetapi model yang **error-nya bisa dijelaskan, kestabilannya bisa diuji, dan performanya tetap masuk akal pada data yang belum pernah dilihat**.

Tantangan kecil untuk pembaca: modifikasi file `bab20_evaluasi_regresi_terpadu.py` agar selain `LinearRegression`, Anda juga menguji `Ridge` dan `Lasso`, lalu bandingkan **MAE, RMSE, R^2, dan mean cross-validation R^2** ketiganya.

Bab 21. Supervised Learning II — Klasifikasi Dasar

Bab ini adalah lanjutan alami setelah bab regresi. Jika pada regresi kita memprediksi **angka**, maka pada klasifikasi kita memprediksi **kelas atau label**. Di outline naskah utama, bab klasifikasi memang dirancang berisi **Logistic Regression, K-Nearest Neighbors, Decision Tree, dan Random Forest**, lengkap dengan latihan kode *step-by-step* menggunakan scikit-learn. Arah ini juga sangat konsisten dengan dokumen lampiran yang membedakan masalah **regression vs classification**, serta menempatkan keempat algoritma ini sebagai fondasi praktis untuk pemula sebelum masuk ke model yang lebih kompleks.

Materi lampiran juga memberi penguatan yang jelas tentang karakter tiap algoritma. **Logistic Regression** diposisikan sebagai model klasifikasi biner yang sederhana, efisien, dan interpretatif. **KNN** dijelaskan sebagai algoritma yang intuitif dan berbasis kedekatan jarak. **Decision Tree** unggul karena mudah dijelaskan melalui aturan keputusan. **Random Forest** memperkuat pohon tunggal lewat *ensemble* banyak pohon, sehingga umumnya lebih stabil dan lebih tahan terhadap *overfitting*. Semua ini sangat cocok untuk pembaca pemula yang ingin memahami klasifikasi secara bertahap, praktis, dan tidak menakutkan.

Bab ini dirancang agar pembaca mampu:

- memahami apa itu klasifikasi
- membedakan **Logistic Regression**, **KNN**, **Decision Tree**, dan **Random Forest**
- mengetahui kapan masing-masing model masuk akal dipakai
- membaca kelebihan dan kekurangannya
- menjalankan **contoh source code Python lengkap**

21.1 Apa itu klasifikasi

Klasifikasi adalah tugas *supervised learning* ketika target yang ingin diprediksi berupa **kelas**, bukan angka kontinu. Contoh paling sederhana:

- lulus / tidak lulus
- spam / tidak spam
- fraud / tidak fraud
- pelanggan aktif / tidak aktif
- penyakit A / bukan penyakit A

Dokumen lampiran menjelaskan bahwa dalam klasifikasi, model belajar dari data berlabel, lalu mencoba memetakan fitur ke kelas yang benar pada data baru. Dalam konteks ini, *Logistic Regression*, *Decision Tree*, dan *KNN* sering dijadikan titik mulai yang sangat baik untuk pembelajaran dan baseline.

Intuisi sederhana

> Kalau regresi menjawab:
>
> **“berapa nilainya?”**

maka klasifikasi menjawab:

“masuk kelas yang mana?”

21.2 Dataset contoh untuk bab ini

Agar pembahasan lebih hidup, kita akan memakai dataset sederhana bertema **kelulusan mahasiswa**.

Simpan sebagai:

```
data/raw/klasifikasi_mahasiswa.csv
```

Isi file:

```
jam_belajar,kehadiran,tugas,lulus
2,70,60,0
3,72,62,0
4,75,65,0
5,78,68,0
6,80,72,1
7,84,76,1
8,87,80,1
9,90,84,1
10,92,87,1
11,95,90,1
4,74,64,0
5,77,67,0
6,81,73,1
7,85,77,1
8,88,81,1
9,91,85,1
10,93,88,1
11,96,91,1
```

Pada dataset ini:

- fitur: `jam_belajar`, `kehadiran`, `tugas`
- target: `lulus`
    - `0` = tidak lulus
    - `1` = lulus

21.3 Metrik evaluasi klasifikasi yang akan dipakai

Sebelum masuk ke model, pembaca perlu tahu bagaimana model klasifikasi dinilai. Dokumen lampiran menekankan pentingnya **accuracy, precision, recall, F1-score, confusion matrix**, dan dalam konteks tertentu juga ROC/AUC. Untuk bab dasar ini, kita fokus pada metrik yang paling mudah dipahami lebih dulu.

Metrik dasar

- **Accuracy**
  proporsi prediksi yang benar
- **Precision**
  dari semua yang diprediksi positif, berapa yang benar-benar positif
- **Recall**
  dari semua yang benar-benar positif, berapa yang berhasil ditemukan model
- **F1-score**
  keseimbangan antara precision dan recall
- **Confusion matrix**
  ringkasan prediksi benar dan salah per kelas

Fungsi evaluasi umum

Simpan sebagai:

```
src/bab21_fungsi_evaluasi_klasifikasi.py

from sklearn.metrics import accuracy_score, confusion_matrix,
classification_report

def evaluasi_klasifikasi(y_test, y_pred, judul="HASIL
EVALUASI"):
    print("=" * 60)
    print(judul)
    print("=" * 60)
    print("Accuracy:", accuracy_score(y_test, y_pred))
    print("\nConfusion Matrix:")
    print(confusion_matrix(y_test, y_pred))
    print("\nClassification Report:")
    print(classification_report(y_test, y_pred))
```

Fungsi ini akan dipakai berulang agar alur evaluasi tetap konsisten.

21.4 Logistic Regression

Intuisi

Walaupun namanya “regression”, **Logistic Regression** dipakai untuk **klasifikasi**, terutama klasifikasi biner. Dokumen lampiran secara eksplisit menekankan bahwa logistic regression menyeimbangkan **simplicity, interpretability, and efficacy** untuk masalah klasifikasi biner, dan sering sangat layak dijadikan baseline pertama.

Kapan memakai Logistic Regression

Gunakan **Logistic Regression** jika:

- target biner
- ingin baseline kuat dan mudah dijelaskan
- ingin probabilitas kelas
- hubungan antarfitur dan target belum terlalu kompleks
- interpretasi lebih penting daripada model yang sangat rumit

Kelebihan

- sederhana
- cepat
- cukup kuat untuk baseline
- mudah dijelaskan
- bisa memberi probabilitas

Kekurangan

- cenderung bekerja lebih baik jika pola pemisahan kelas tidak terlalu rumit
- bisa kalah dari model nonlinier pada data tabular kompleks
- sering lebih baik jika fitur diskalakan dengan rapi

Source code Python lengkap

Simpan sebagai:

```
src/bab21_logistic_regression.py

import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import accuracy_score, confusion_matrix,
classification_report

# 1. Load data
df = pd.read_csv("data/raw/klasifikasi_mahasiswa.csv")

# 2. Fitur dan target
X = df[["jam_belajar", "kehadiran", "tugas"]]
y = df["lulus"]

# 3. Split data
X_train, X_test, y_train, y_test = train_test_split(
    X, y, test_size=0.25, random_state=42
)

# 4. Scaling
scaler = StandardScaler()
X_train_scaled = scaler.fit_transform(X_train)
X_test_scaled = scaler.transform(X_test)

# 5. Model
```

```
model = LogisticRegression()
model.fit(X_train_scaled, y_train)

# 6. Prediksi
y_pred = model.predict(X_test_scaled)
y_prob = model.predict_proba(X_test_scaled)

# 7. Evaluasi
print("=== LOGISTIC REGRESSION ===")
print("Accuracy:", accuracy_score(y_test, y_pred))
print("\nConfusion Matrix:")
print(confusion_matrix(y_test, y_pred))
print("\nClassification Report:")
print(classification_report(y_test, y_pred))

hasil = pd.DataFrame({
    "aktual": y_test.values,
    "prediksi": y_pred,
    "prob_tidak_lulus": y_prob[:, 0],
    "prob_lulus": y_prob[:, 1]
})

print("\nHasil prediksi:")
print(hasil)
```

Apa yang dipelajari

Dari model ini, pembaca belajar bahwa klasifikasi tidak hanya menghasilkan kelas `0` atau `1`, tetapi juga **probabilitas** untuk tiap kelas. Ini sangat berguna dalam pengambilan keputusan.

21.5 K-Nearest Neighbors (KNN)

Intuisi

KNN adalah salah satu algoritma klasifikasi paling intuitif. Dokumen lampiran menjelaskan bahwa KNN bekerja berdasarkan prinsip **kedekatan**: data yang letaknya berdekatan cenderung memiliki kelas yang sama. Saat ada data baru, KNN melihat `k` tetangga terdekat, lalu mengambil keputusan berdasarkan suara mayoritas.

Kapan memakai KNN

Gunakan **KNN** jika:

- ingin baseline cepat
- dataset tidak terlalu besar
- fitur numerik bisa di-*scale*
- ingin model yang sangat mudah dipahami secara konsep

Kelebihan

- konsep sederhana
- cepat dipahami
- sering bagus sebagai baseline
- tidak butuh proses training rumit

Kekurangan

- sensitif terhadap scaling
- sensitif terhadap pemilihan `k`
- bisa lambat saat prediksi jika data besar
- bisa lemah jika fitur banyak dan ruang fitur menjadi renggang

Kenapa scaling penting

Karena KNN bergantung pada jarak, fitur dengan skala besar bisa mendominasi fitur lain. Itu sebabnya preprocessing seperti `StandardScaler` sangat penting untuk KNN. Ini juga sangat selaras dengan praktik lampiran yang menekankan pentingnya scaling pada model berbasis jarak.

Source code Python lengkap

Simpan sebagai:

```
src/bab21_knn.py

import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.neighbors import KNeighborsClassifier
from sklearn.metrics import accuracy_score, confusion_matrix,
classification_report

# 1. Load data
df = pd.read_csv("data/raw/klasifikasi_mahasiswa.csv")

# 2. Fitur dan target
X = df[["jam_belajar", "kehadiran", "tugas"]]
y = df["lulus"]

# 3. Split data
X_train, X_test, y_train, y_test = train_test_split(
    X, y, test_size=0.25, random_state=42
)

# 4. Scaling
scaler = StandardScaler()
X_train_scaled = scaler.fit_transform(X_train)
X_test_scaled = scaler.transform(X_test)
```

```
# 5. Model
model = KNeighborsClassifier(n_neighbors=3)
model.fit(X_train_scaled, y_train)

# 6. Prediksi
y_pred = model.predict(X_test_scaled)

# 7. Evaluasi
print("=== K-NEAREST NEIGHBORS ===")
print("Accuracy:", accuracy_score(y_test, y_pred))
print("\nConfusion Matrix:")
print(confusion_matrix(y_test, y_pred))
print("\nClassification Report:")
print(classification_report(y_test, y_pred))
```

Apa yang dipelajari

KNN sangat bagus untuk membantu pembaca melihat bahwa klasifikasi bisa dilakukan tanpa “persamaan rumit”, cukup lewat logika kedekatan.

21.6 Decision Tree

Intuisi

Decision Tree bekerja seperti serangkaian pertanyaan keputusan. Dokumen lampiran menjelaskan pohon keputusan sebagai struktur mirip *flowchart*: setiap node internal adalah tes terhadap fitur, setiap cabang adalah hasil tes itu, dan setiap daun adalah keputusan kelas. Karena sifatnya yang sangat visual, Decision Tree adalah salah satu model yang paling mudah dijelaskan ke orang nonteknis.

Kapan memakai Decision Tree

Gunakan **Decision Tree** jika:

- ingin model mudah dijelaskan
- ingin baseline cepat
- ingin model yang tidak terlalu menuntut scaling
- ingin melihat aturan keputusan secara eksplisit

Kelebihan

- sangat interpretatif
- mudah divisualisasikan
- tidak terlalu butuh scaling
- bisa menangkap hubungan nonlinier sederhana

Kekurangan

- mudah *overfit*
- bisa tidak stabil terhadap perubahan kecil pada data

- satu pohon kadang kalah kuat dari model ensemble

Dokumen lampiran juga mengingatkan bahwa pohon keputusan rawan overfitting, sehingga pruning atau pembatasan kedalaman pohon menjadi penting.

Source code Python lengkap

Simpan sebagai:

```
src/bab21_decision_tree.py

import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.tree import DecisionTreeClassifier, export_text
from sklearn.metrics import accuracy_score, confusion_matrix,
classification_report

# 1. Load data
df = pd.read_csv("data/raw/klasifikasi_mahasiswa.csv")

# 2. Fitur dan target
X = df[["jam_belajar", "kehadiran", "tugas"]]
y = df["lulus"]

# 3. Split data
X_train, X_test, y_train, y_test = train_test_split(
    X, y, test_size=0.25, random_state=42
)

# 4. Model
model = DecisionTreeClassifier(max_depth=3, random_state=42)
model.fit(X_train, y_train)

# 5. Prediksi
y_pred = model.predict(X_test)

# 6. Evaluasi
print("=== DECISION TREE ===")
print("Accuracy:", accuracy_score(y_test, y_pred))
print("\nConfusion Matrix:")
print(confusion_matrix(y_test, y_pred))
print("\nClassification Report:")
print(classification_report(y_test, y_pred))

# 7. Aturan pohon
aturan = export_text(model, feature_names=list(X.columns))
print("\nAturan Decision Tree:")
print(aturan)
```

Apa yang dipelajari

Decision Tree membantu pembaca memahami bahwa model bisa dipandang sebagai **aturan keputusan**, bukan selalu persamaan matematis.

21.7 Random Forest

Intuisi

Kalau Decision Tree adalah satu pohon, maka **Random Forest** adalah kumpulan banyak pohon keputusan yang bekerja bersama. Dokumen lampiran menekankan bahwa Random Forest memakai *bagging* dan subset fitur acak untuk menghasilkan model yang lebih kuat, lebih stabil, dan lebih tahan terhadap overfitting dibanding pohon tunggal. Ia juga sering memberi **feature importance**, yang sangat berguna dalam analisis tabular.

Kapan memakai Random Forest

Gunakan **Random Forest** jika:

- klasifikasi tabular cukup kompleks
- baseline sederhana mulai terasa kurang kuat
- ingin model yang umumnya stabil dan kuat
- interpretasi total tidak harus sesederhana Logistic Regression atau Decision Tree tunggal

Kelebihan

- kuat untuk data tabular
- lebih tahan terhadap overfitting daripada satu Decision Tree
- cukup robust terhadap noise
- menyediakan *feature importance*

Kekurangan

- lebih sulit dijelaskan daripada satu pohon
- lebih berat secara komputasi
- tidak sesederhana Logistic Regression untuk interpretasi langsung

Source code Python lengkap

Simpan sebagai:

src/bab21_random_forest.py

```
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import accuracy_score, confusion_matrix,
classification_report

# 1. Load data
df = pd.read_csv("data/raw/klasifikasi_mahasiswa.csv")

# 2. Fitur dan target
X = df[["jam_belajar", "kehadiran", "tugas"]]
y = df["lulus"]

# 3. Split data
X_train, X_test, y_train, y_test = train_test_split(
    X, y, test_size=0.25, random_state=42
)

# 4. Model
model = RandomForestClassifier(random_state=42)
model.fit(X_train, y_train)

# 5. Prediksi
y_pred = model.predict(X_test)

# 6. Evaluasi
print("=== RANDOM FOREST ===")
print("Accuracy:", accuracy_score(y_test, y_pred))
print("\nConfusion Matrix:")
print(confusion_matrix(y_test, y_pred))
print("\nClassification Report:")
print(classification_report(y_test, y_pred))

# 7. Feature importance
importances = pd.DataFrame({
    "fitur": X.columns,
    "importance": model.feature_importances_
}).sort_values(by="importance", ascending=False)

print("\nFeature Importance:")
print(importances)
```

Apa yang dipelajari

Random Forest membantu pembaca melihat bahwa model ensemble bisa meningkatkan kekuatan prediksi tanpa harus langsung masuk ke algoritma yang sangat kompleks.

21.8 Perbandingan kelebihan dan kekurangan

Bagian ini penting agar pembaca tidak hanya melihat kode, tetapi juga memahami **kompromi** antar model.

Logistic Regression

Kelebihan

- sederhana
- cepat
- cukup kuat sebagai baseline
- mudah dijelaskan
- memberi probabilitas kelas

Kekurangan

- lebih cocok untuk pemisahan kelas yang relatif sederhana
- bisa kalah jika pola data non linier dan kompleks

KNN

Kelebihan

- sangat intuitif
- baseline cepat
- tidak butuh training rumit

Kekurangan

- sensitif terhadap scaling
- sensitif terhadap pemilihan k
- bisa lambat saat data membesar

Decision Tree

Kelebihan

- sangat mudah dijelaskan
- bisa menghasilkan aturan keputusan
- tidak terlalu menuntut scaling
- menangkap nonlinieritas sederhana

Kekurangan

- mudah overfit
- satu pohon bisa tidak stabil

Random Forest

Kelebihan

- kuat untuk data tabular
- lebih stabil dari satu pohon
- robust terhadap noise
- memberi *feature importance*

Kekurangan

- lebih berat
- interpretasi lebih sulit daripada model sederhana

Dokumen lampiran sangat konsisten dengan pembacaan ini: Logistic Regression dipuji karena keseimbangan antara kesederhanaan dan efektivitas, KNN karena intuitif, Decision Tree karena interpretabilitas, dan Random Forest karena kombinasi kekuatan prediktif dan robustnes.

21.9 Kapan memakai masing-masing

Agar mudah diingat, pakai aturan praktis berikut.

Mulai dari Logistic Regression jika:

- target biner
- ingin model baseline yang rapi
- ingin model mudah dijelaskan
- ingin probabilitas kelas

Coba KNN jika:

- ingin baseline berbasis kedekatan
- fitur numerik sudah diskalakan
- data tidak terlalu besar

Coba Decision Tree jika:

- ingin aturan keputusan yang eksplisit
- ingin model cepat dan mudah divisualisasikan
- ingin baseline yang tidak terlalu repot preprocessing

Coba Random Forest jika:

- data tabular mulai terasa lebih kompleks
- ingin performa yang biasanya lebih kuat
- ingin model lebih tahan terhadap overfitting daripada satu tree

21.10 Source code Python lengkap: perbandingan empat model

Berikut satu file yang membandingkan keempat model sekaligus.

Simpan sebagai:

```
src/bab21_perbandingan_klasifikasi.py

import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.linear_model import LogisticRegression
from sklearn.neighbors import KNeighborsClassifier
from sklearn.tree import DecisionTreeClassifier
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import accuracy_score

# Load data
df = pd.read_csv("data/raw/klasifikasi_mahasiswa.csv")

X = df[["jam_belajar", "kehadiran", "tugas"]]
y = df["lulus"]

X_train, X_test, y_train, y_test = train_test_split(
    X, y, test_size=0.25, random_state=42
)

# Scaling khusus untuk model berbasis jarak / model yang terbantu
scaling
scaler = StandardScaler()
X_train_scaled = scaler.fit_transform(X_train)
X_test_scaled = scaler.transform(X_test)

models = {
    "LogisticRegression": LogisticRegression(),
    "KNN": KNeighborsClassifier(n_neighbors=3),
    "DecisionTree": DecisionTreeClassifier(max_depth=3,
random_state=42),
    "RandomForest": RandomForestClassifier(random_state=42)
}

print("=" * 60)
print("PERBANDINGAN MODEL KLASIFIKASI DASAR")
print("=" * 60)

for nama, model in models.items():
```

```
    if nama in ["LogisticRegression", "KNN"]:
        model.fit(X_train_scaled, y_train)
        y_pred = model.predict(X_test_scaled)
    else:
        model.fit(X_train, y_train)
        y_pred = model.predict(X_test)

    acc = accuracy_score(y_test, y_pred)
    print(f"{nama}: Accuracy = {acc:.4f}")
```

Program ini sangat bagus sebagai latihan ringkas untuk melihat bahwa model berbeda bisa memberi hasil berbeda pada dataset yang sama.

21.11 Alur kerja yang disarankan untuk pembaca

Agar pembelajaran lebih sehat, saya sarankan urutan berikut:

- mulai dari **Logistic Regression** sebagai baseline klasifikasi biner
- bandingkan dengan **KNN**
- coba **Decision Tree** untuk melihat aturan keputusan
- lanjut ke **Random Forest** untuk melihat peningkatan stabilitas dan kekuatan prediksi
- bandingkan hasil evaluasi
- jangan hanya membaca accuracy; lihat juga confusion matrix dan classification report

Ini sangat sesuai dengan semangat materi lampiran yang menekankan *workflow* konsisten, evaluasi yang benar, dan perbandingan model pada data yang sama.

21.12 Kesalahan umum pemula

Ada beberapa jebakan yang sering muncul.

- **mengira accuracy saja sudah cukup**
  padahal precision, recall, dan confusion matrix bisa bercerita lebih banyak
- **tidak melakukan scaling untuk KNN**
  sehingga hasil model menjadi bias oleh skala fitur
- **menganggap Random Forest pasti selalu terbaik**
  padahal tergantung data, target, dan kebutuhan interpretasi
- **langsung memilih model kompleks tanpa baseline**
  padahal Logistic Regression sering sudah cukup kuat sebagai titik awal
- **tidak membatasi kedalaman Decision Tree**
  sehingga model mudah overfit

Dokumen lampiran juga berulang kali mengingatkan pentingnya membaca metrik klasifikasi secara lengkap dan berhati-hati terhadap overfitting.

21.13 Refleksi: klasifikasi dasar adalah laboratorium yang sangat baik

Bab ini penting karena klasifikasi dasar mengajarkan banyak konsep inti sekaligus:

- label dan kelas
- probabilitas kelas
- confusion matrix
- precision dan recall
- interpretabilitas vs performa
- baseline vs model ensemble

Itulah mengapa empat model ini sangat cocok diletakkan bersama. Mereka memberi pembaca spektrum yang jelas:

- **Logistic Regression** → sederhana dan rapi
- **KNN** → intuitif
- **Decision Tree** → mudah dijelaskan
- **Random Forest** → lebih kuat dan stabil

Dengan menguasai empat model ini, pembaca akan lebih siap masuk ke klasifikasi yang lebih lanjut, tuning model, dan evaluasi yang lebih kaya. Ini juga sangat selaras dengan semangat outline utama yang menempatkan klasifikasi dasar sebagai inti pembelajaran ML pemula.

Penutup Bab

Bab ini membahas **Supervised Learning II — Klasifikasi Dasar** melalui empat model penting: **Logistic Regression, K-Nearest Neighbors, Decision Tree, dan Random Forest**, lengkap dengan **perbandingan kelebihan-kekurangan** serta **contoh source code Python**. Seluruh pembahasan ini sangat relevan dengan dokumen lampiran yang menekankan peran Logistic Regression sebagai baseline klasifikasi biner yang interpretatif, KNN sebagai model berbasis kedekatan yang intuitif, Decision Tree sebagai model yang mudah dijelaskan, dan Random Forest sebagai ensemble yang lebih kuat dan lebih tahan terhadap overfitting.

Pesan terpenting dari bab ini: pada klasifikasi, model yang baik bukan hanya yang akurasinya tinggi, tetapi yang **sesuai dengan data, tujuan, dan kebutuhan interpretasi**.

Tantangan kecil untuk pembaca: modifikasi file `bab21_perbandingan_klasifikasi.py` agar selain accuracy, program juga menampilkan **precision, recall, dan F1-score** untuk keempat model, lalu lihat apakah model dengan accuracy tertinggi juga selalu paling seimbang performanya.

Bab 22. Evaluasi Model Klasifikasi

Bab ini sangat penting karena pada titik inilah pembaca belajar bahwa **model klasifikasi yang "jalan" belum tentu model yang benar-benar berguna**. Di outline naskah utama, Bab 22 memang berfokus pada **confusion matrix, accuracy, precision, recall, F1-score, ROC-AUC secara intuitif, dan class imbalance dasar**. Susunan ini sangat selaras dengan materi lampiran yang menekankan bahwa evaluasi model klasifikasi tidak boleh berhenti pada satu angka, karena setiap metrik memberi sudut pandang berbeda tentang performa model. Accuracy memberi gambaran umum, precision dan recall menyoroti jenis kesalahan yang berbeda, F1-score membantu menyeimbangkan keduanya, sedangkan ROC-AUC memberi pandangan lintas ambang keputusan. Materi lampiran juga menegaskan bahwa pada data yang tidak seimbang, accuracy saja bisa sangat menyesatkan.

Referensi lampiran juga memperkuat posisi tiap metrik. **Confusion matrix** dijelaskan sebagai tabel yang memecah prediksi benar dan salah ke dalam kombinasi aktual versus prediksi, sehingga sangat berguna untuk membaca performa model lebih dalam daripada sekadar accuracy. **Precision** mengukur ketepatan prediksi positif, **recall** mengukur kemampuan menangkap seluruh kasus positif, dan **F1-score** menjadi ringkasan seimbang keduanya, terutama ketika distribusi kelas miring. Pada sisi ROC-AUC, lampiran menjelaskan bahwa kurva ROC memetakan *true positive rate* terhadap *false positive rate* di berbagai threshold, sementara AUC merangkum kualitas diskriminasi model; namun untuk dataset yang sangat tidak seimbang, ROC-AUC juga perlu dibaca hati-hati karena bisa tampak terlalu optimistis.

Bab ini dirancang agar pembaca mampu:

- memahami **confusion matrix**
- membaca **accuracy**
- membaca **precision**
- membaca **recall**
- membaca **F1-score**
- memahami **ROC-AUC secara intuitif**
- mengenali **class imbalance** dasar
- menjalankan **source code Python lengkap** untuk evaluasi model klasifikasi

22.1 Mengapa evaluasi model klasifikasi tidak boleh hanya memakai accuracy

Banyak pemula senang ketika melihat accuracy tinggi, tetapi materi lampiran justru menegaskan bahwa **accuracy bisa menyesatkan**, terutama bila satu kelas jauh lebih dominan daripada kelas lain. Referensi Van Der Post memberi contoh kasus *fraud detection* atau diagnosis penyakit langka: model yang selalu menebak kelas mayoritas bisa mendapat accuracy tinggi, padahal gagal total dalam menemukan kasus penting dari kelas minoritas. Saltz juga menekankan bahwa accuracy harus ditafsirkan dalam konteks masalah, dan bahwa laporan performa klasifikasi sebaiknya selalu melampaui accuracy saja.

Secara praktis, model klasifikasi perlu menjawab lebih dari satu pertanyaan:

- berapa banyak prediksi yang benar secara keseluruhan
- berapa banyak prediksi positif yang benar
- berapa banyak kasus positif yang berhasil ditemukan
- apakah model seimbang antara menghindari *false positive* dan *false negative*
- bagaimana performanya jika threshold diubah

Karena itu, evaluasi klasifikasi harus dibaca sebagai **paket metrik**, bukan satu angka tunggal.

22.2 Dataset contoh untuk bab ini

Agar pembahasan lebih hidup, kita akan memakai dataset sederhana bertema kelulusan mahasiswa.

Simpan sebagai:

```
data/raw/evaluasi_klasifikasi_mahasiswa.csv
```

Isi file:

```
jam_belajar,kehadiran,tugas,lulus
2,70,60,0
3,72,62,0
4,75,65,0
5,78,68,0
6,80,72,1
7,84,76,1
8,87,80,1
9,90,84,1
10,92,87,1
11,95,90,1
4,74,64,0
5,77,67,0
6,81,73,1
7,85,77,1
8,88,81,1
9,91,85,1
10,93,88,1
11,96,91,1
```

Pada dataset ini:

- fitur: `jam_belajar`, `kehadiran`, `tugas`
- target: `lulus`
    - `0` = tidak lulus
    - `1` = lulus

Dataset ini cukup baik untuk memperkenalkan evaluasi klasifikasi dasar dengan **Logistic Regression** sebagai model awal, lalu membaca semua metrik yang akan dibahas.

22.3 Menyiapkan model awal untuk dievaluasi

Sebelum menghitung metrik, kita tentu perlu melatih model lebih dulu. Untuk bab ini, kita pakai **Logistic Regression** sebagai model dasar, karena lampiran menempatkannya sebagai baseline klasifikasi biner yang sederhana, efisien, dan cukup interpretatif.

Simpan sebagai:

```
src/bab22_model_awal_klasifikasi.py

import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.linear_model import LogisticRegression

# 1. Load data
df = pd.read_csv("data/raw/evaluasi_klasifikasi_mahasiswa.csv")

# 2. Fitur dan target
X = df[["jam_belajar", "kehadiran", "tugas"]]
y = df["lulus"]

# 3. Split data
X_train, X_test, y_train, y_test = train_test_split(
    X, y,
    test_size=0.25,
    random_state=42
)

# 4. Scaling
scaler = StandardScaler()
X_train_scaled = scaler.fit_transform(X_train)
X_test_scaled = scaler.transform(X_test)

# 5. Model
model = LogisticRegression()
model.fit(X_train_scaled, y_train)

# 6. Prediksi
y_pred = model.predict(X_test_scaled)
y_prob = model.predict_proba(X_test_scaled)[:, 1]

hasil = pd.DataFrame({
    "aktual": y_test.values,
    "prediksi": y_pred,
    "prob_lulus": y_prob
})

print(hasil)
```

Program ini menjadi fondasi evaluasi pada bagian-bagian berikut.

22.4 Confusion matrix

Apa itu confusion matrix

Confusion matrix adalah tabel yang menunjukkan jumlah prediksi untuk setiap kombinasi antara kelas aktual dan kelas prediksi. Saltz menjelaskannya sebagai tabel yang memuat seluruh kemungkinan prediksi benar dan salah, sehingga kita bisa melihat bukan hanya seberapa banyak model benar, tetapi **di mana** model salah. Referensi Lim juga menegaskan bahwa pada klasifikasi multi-kelas maupun biner, confusion matrix dibaca per baris sebagai kelas aktual dan per kolom sebagai kelas prediksi.

Pada klasifikasi biner, confusion matrix berisi empat elemen utama:

- **True Positive (TP)** → aktual positif, diprediksi positif
- **True Negative (TN)** → aktual negatif, diprediksi negatif
- **False Positive (FP)** → aktual negatif, diprediksi positif
- **False Negative (FN)** → aktual positif, diprediksi negatif

Referensi Van Der Post juga menekankan bahwa confusion matrix memberi wawasan yang jauh lebih kaya daripada accuracy tunggal, terutama pada masalah yang sensitif terhadap jenis error tertentu.

Source code Python lengkap

Simpan sebagai:

```
src/bab22_confusion_matrix.py

import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import confusion_matrix

df = pd.read_csv("data/raw/evaluasi_klasifikasi_mahasiswa.csv")

X = df[["jam_belajar", "kehadiran", "tugas"]]
y = df["lulus"]

X_train, X_test, y_train, y_test = train_test_split(
    X, y,
    test_size=0.25,
    random_state=42
)

scaler = StandardScaler()
X_train_scaled = scaler.fit_transform(X_train)
X_test_scaled = scaler.transform(X_test)

model = LogisticRegression()
model.fit(X_train_scaled, y_train)
```

```
y_pred = model.predict(X_test_scaled)

cm = confusion_matrix(y_test, y_pred)

print("=== CONFUSION MATRIX ===")
print(cm)
```

Cara membaca confusion matrix

Kalau hasilnya seperti:

```
[[2 0]
 [1 2]]
```

artinya:

- 2 data tidak lulus diprediksi benar sebagai tidak lulus
- 0 data tidak lulus salah diprediksi lulus
- 1 data lulus salah diprediksi tidak lulus
- 2 data lulus diprediksi benar sebagai lulus

Dengan confusion matrix, pembaca mulai bisa melihat **jenis kesalahan model**, bukan hanya jumlah totalnya. Ini sangat penting untuk membaca precision, recall, dan F1-score.

22.5 Accuracy

Apa itu accuracy

Accuracy adalah rasio total prediksi yang benar terhadap seluruh data uji. Saltz menyebut accuracy sebagai metrik paling langsung dan paling mudah dipahami, tetapi juga mengingatkan bahwa accuracy harus dibaca dalam konteks masalah. Van Der Post menambahkan bahwa accuracy bisa sangat menipu pada dataset yang tidak seimbang.

Rumus intuisinya:

Accuracy = (TP + TN) / seluruh observasi

Kapan accuracy berguna

Accuracy cukup berguna jika:

- distribusi kelas relatif seimbang
- biaya salah prediksi antar kelas tidak terlalu berbeda
- Anda butuh gambaran umum awal

Source code Python lengkap

Simpan sebagai:

```
src/bab22_accuracy.py

import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import accuracy_score

df = pd.read_csv("data/raw/evaluasi_klasifikasi_mahasiswa.csv")

X = df[["jam_belajar", "kehadiran", "tugas"]]
y = df["lulus"]

X_train, X_test, y_train, y_test = train_test_split(
    X, y,
    test_size=0.25,
    random_state=42
)

scaler = StandardScaler()
X_train_scaled = scaler.fit_transform(X_train)
X_test_scaled = scaler.transform(X_test)

model = LogisticRegression()
model.fit(X_train_scaled, y_train)

y_pred = model.predict(X_test_scaled)

acc = accuracy_score(y_test, y_pred)

print("Accuracy:", acc)
```

Cara membaca accuracy

Kalau accuracy = 0.80, artinya:

> **80% data uji diklasifikasikan dengan benar**

Tetapi seperti ditekankan di lampiran, accuracy **tidak cukup** jika satu kelas sangat dominan atau jika *false negative* jauh lebih mahal daripada *false positive*.

22.6 Precision

Apa itu precision

Precision adalah rasio antara prediksi positif yang benar terhadap seluruh prediksi positif. Saltz menjelaskan precision sebagai fokus pada kualitas prediksi positif: dari semua yang diprediksi sebagai kelas 1, berapa yang benar-benar 1. Uddin juga menekankan bahwa precision menjawab pertanyaan: **berapa banyak item yang dipilih model sebagai positif memang relevan?**

Rumus intuisinya:

Precision = TP / (TP + FP)

Kapan precision penting

Precision penting jika:

- *false positive* mahal
- kita ingin prediksi positif benar-benar dapat dipercaya
- contoh: alarm fraud, diagnosis, sistem rekomendasi intervensi

Van Der Post memberi contoh medis: precision penting agar orang sehat tidak terlalu sering salah ditandai sakit.

Source code Python lengkap

Simpan sebagai:

```
src/bab22_precision.py

import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import precision_score

df = pd.read_csv("data/raw/evaluasi_klasifikasi_mahasiswa.csv")

X = df[["jam_belajar", "kehadiran", "tugas"]]
y = df["lulus"]

X_train, X_test, y_train, y_test = train_test_split(
    X, y,
    test_size=0.25,
    random_state=42
)

scaler = StandardScaler()
X_train_scaled = scaler.fit_transform(X_train)
X_test_scaled = scaler.transform(X_test)

model = LogisticRegression()
model.fit(X_train_scaled, y_train)

y_pred = model.predict(X_test_scaled)

precision = precision_score(y_test, y_pred)

print("Precision:", precision)
```

Cara membaca precision

Kalau precision = 0.90, artinya:

dari semua yang diprediksi lulus, 90% memang benar-benar lulus

22.7 Recall

Apa itu recall

Recall adalah rasio antara seluruh kasus positif yang berhasil ditemukan model terhadap seluruh kasus positif aktual. Saltz menjelaskan recall sebagai ukuran keberhasilan menangkap seluruh kasus positif, dan mencatat bahwa metrik ini juga sering disebut **sensitivity**. Uddin juga menekankan bahwa recall menjawab: **seberapa baik model menemukan seluruh contoh yang relevan?**

Rumus intuisinya:

Recall = TP / (TP + FN)

Kapan recall penting

Recall penting jika:

- *false negative* mahal
- kita tidak ingin banyak kasus positif lolos
- contoh: deteksi penyakit, fraud, kejadian langka

Van Der Post menekankan contoh yang sama: pada diagnosis penyakit, recall tinggi penting agar sedikit mungkin kasus positif terlewat.

Source code Python lengkap

Simpan sebagai:

```
src/bab22_recall.py

import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import recall_score

df = pd.read_csv("data/raw/evaluasi_klasifikasi_mahasiswa.csv")

X = df[["jam_belajar", "kehadiran", "tugas"]]
y = df["lulus"]

X_train, X_test, y_train, y_test = train_test_split(
    X, y,
    test_size=0.25,
    random_state=42
)

scaler = StandardScaler()
X_train_scaled = scaler.fit_transform(X_train)
X_test_scaled = scaler.transform(X_test)

model = LogisticRegression()
model.fit(X_train_scaled, y_train)

y_pred = model.predict(X_test_scaled)
recall = recall_score(y_test, y_pred)
print("Recall:", recall)
```

Cara membaca recall

Kalau recall = 0.85, artinya:

> **dari semua mahasiswa yang benar-benar lulus, 85% berhasil dikenali model sebagai lulus**

22.8 F1-score

Apa itu F1-score

F1-score adalah rata-rata harmonik dari precision dan recall. Saltz menjelaskan F1 sebagai ringkasan seimbang dari dua metrik itu, sedangkan Uddin menegaskan bahwa F1 sangat berguna ketika kita ingin model yang bukan hanya tepat saat menandai positif, tetapi juga tidak banyak melewatkan kasus positif. Van Der Post juga menekankan bahwa F1 sangat berguna pada **imbalanced datasets**, karena ia menghindari rasa aman palsu dari accuracy tinggi.

Kapan F1-score penting

F1-score penting jika:

- precision dan recall sama-sama penting
- dataset tidak seimbang
- kita ingin satu angka ringkas yang lebih adil daripada accuracy

Source code Python lengkap

Simpan sebagai:

```
src/bab22_f1_score.py

import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import f1_score

df = pd.read_csv("data/raw/evaluasi_klasifikasi_mahasiswa.csv")

X = df[["jam_belajar", "kehadiran", "tugas"]]
y = df["lulus"]

X_train, X_test, y_train, y_test = train_test_split(
    X, y,
    test_size=0.25,
    random_state=42
)

scaler = StandardScaler()
X_train_scaled = scaler.fit_transform(X_train)
X_test_scaled = scaler.transform(X_test)

model = LogisticRegression()
model.fit(X_train_scaled, y_train)

y_pred = model.predict(X_test_scaled)
f1 = f1_score(y_test, y_pred)
print("F1-score:", f1)
```

Cara membaca F1-score

Kalau F1-score tinggi, itu berarti model relatif seimbang antara:

- tidak terlalu banyak *false positive*
- tidak terlalu banyak *false negative*

Ini sebabnya F1 sering dipakai ketika data tidak seimbang atau ketika dua jenis kesalahan sama-sama penting.

22.9 `classification_report()` sebagai ringkasan cepat

Scikit-learn menyediakan `classification_report()` untuk merangkum **precision, recall, F1-score**, dan **support** per kelas. Saltz secara eksplisit membahas bahwa fungsi ini memberi banyak output yang sangat berguna, dan merekomendasikan agar laporan performa klasifikasi memuat accuracy sekaligus precision, recall, dan F1. Referensi Van Der Post juga menampilkan pola evaluasi yang sama pada pipeline klasifikasi di Python.

Source code Python lengkap

Simpan sebagai:

```
src/bab22_classification_report.py

import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import classification_report

df = pd.read_csv("data/raw/evaluasi_klasifikasi_mahasiswa.csv")

X = df[["jam_belajar", "kehadiran", "tugas"]]
y = df["lulus"]

X_train, X_test, y_train, y_test = train_test_split(
    X, y,
    test_size=0.25,
    random_state=42
)

scaler = StandardScaler()
X_train_scaled = scaler.fit_transform(X_train)
X_test_scaled = scaler.transform(X_test)

model = LogisticRegression()
model.fit(X_train_scaled, y_train)

y_pred = model.predict(X_test_scaled)

print(classification_report(y_test, y_pred))
```

Apa itu support

Support menunjukkan jumlah data aktual pada tiap kelas di data uji. Ini sangat penting ketika membaca class imbalance, karena dua model dengan accuracy mirip bisa sangat berbeda kualitasnya jika support antar kelas tidak seimbang. Saltz menegaskan bahwa metrik harus dibaca dalam konteks distribusi kelas.

22.10 ROC-AUC secara intuitif

A. Apa itu ROC

ROC (*Receiver Operating Characteristic*) curve adalah kurva yang memetakan *true positive rate* terhadap *false positive rate* di berbagai threshold. Referensi Knox menjelaskan bahwa setiap titik pada kurva ROC merepresentasikan satu threshold tertentu, dan kurva ini memperlihatkan *trade-off* antara dua jenis error ketika threshold diubah. Van Der Post juga menegaskan bahwa ROC adalah alat visual untuk melihat performa classifier di berbagai threshold keputusan.

B. Apa itu AUC

AUC (*Area Under the Curve*) merangkum kualitas diskriminasi model. Lampiran menjelaskan bahwa:

- **AUC = 1.0** berarti klasifikasi sempurna
- **AUC = 0.5** berarti tidak lebih baik dari tebakan acak

ROC-AUC sangat berguna untuk melihat kemampuan model membedakan dua kelas secara umum, bukan hanya pada satu threshold tetap.

C. Intuisi sederhana

Bayangkan model memberi probabilitas lulus untuk setiap mahasiswa. Threshold default sering `0.5`, tetapi jika threshold diubah menjadi `0.4` atau `0.7`, hasil TP dan FP akan ikut berubah. ROC membantu kita melihat perilaku model **di seluruh threshold**, bukan hanya satu titik. Knox menekankan bahwa ROC membantu praktisi dan klien memahami titik operasi yang sesuai dengan konteks risiko.

D. Source code Python lengkap

Simpan sebagai:

```
src/bab22_roc_auc_intuitif.py

import pandas as pd
import matplotlib.pyplot as plt
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import roc_curve, roc_auc_score

df = pd.read_csv("data/raw/evaluasi_klasifikasi_mahasiswa.csv")

X = df[["jam_belajar", "kehadiran", "tugas"]]
y = df["lulus"]

X_train, X_test, y_train, y_test = train_test_split(
    X, y,
    test_size=0.25,
    random_state=42
)

scaler = StandardScaler()
X_train_scaled = scaler.fit_transform(X_train)
X_test_scaled = scaler.transform(X_test)

model = LogisticRegression()
model.fit(X_train_scaled, y_train)

y_prob = model.predict_proba(X_test_scaled)[:, 1]

auc = roc_auc_score(y_test, y_prob)
fpr, tpr, thresholds = roc_curve(y_test, y_prob)

print("ROC-AUC:", auc)

plt.figure(figsize=(7, 5))
plt.plot(fpr, tpr)
plt.plot([0, 1], [0, 1], linestyle="--")
plt.xlabel("False Positive Rate")
plt.ylabel("True Positive Rate")
plt.title("ROC Curve")
plt.grid(True)
plt.tight_layout()
plt.show()
```

E. Cara membaca ROC-AUC secara intuitif

- kurva yang lebih dekat ke kiri atas → biasanya lebih baik
- garis diagonal → kira-kira setara tebakan acak
- AUC lebih tinggi → pemisahan kelas lebih baik

F. Catatan penting tentang ROC-AUC

Lampiran juga mengingatkan bahwa pada **dataset yang sangat tidak seimbang**, ROC-AUC bisa tampak terlalu optimistis. Dalam situasi seperti itu, precision-recall perspective sering menjadi pendamping yang baik. Jadi ROC-AUC kuat, tetapi tetap harus dibaca dalam konteks.

22.11 *Class imbalance* dasar

Apa itu class imbalance

Class imbalance terjadi ketika satu kelas jauh lebih banyak daripada kelas lain. Van Der Post menekankan bahwa situasi ini sering muncul pada *fraud detection*, diagnosis penyakit, dan kejadian langka, dan bahwa imbalance dapat membuat model terlalu condong memprediksi kelas mayoritas.

Kenapa ini berbahaya

Kalau 95% data adalah kelas `0`, maka model yang selalu menebak `0` bisa mendapat accuracy 95%, padahal sebenarnya gagal total menemukan kelas `1`. Van Der Post menjelaskan contoh seperti ini secara eksplisit, dan Saltz juga menekankan bahwa metrik harus dibaca sesuai konteks masalah, bukan hanya angka yang terlihat tinggi.

Contoh dataset imbalanced sederhana

Simpan sebagai:

```
data/raw/imbalance_sederhana.csv
```

Isi file:

```
jam_belajar,kehadiran,tugas,lulus
2,70,60,0
3,72,62,0
4,75,65,0
4,74,64,0
5,77,67,0
5,78,68,0
3,71,61,0
4,73,63,0
5,76,66,0
6,80,72,1
```

Di sini kelas `0` jauh lebih banyak daripada kelas `1`.

Source code Python lengkap

Simpan sebagai:

```
src/bab22_class_imbalance_dasar.py

import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import accuracy_score, classification_report,
confusion_matrix

df = pd.read_csv("data/raw/imbalance_sederhana.csv")

print("Distribusi kelas:")
print(df["lulus"].value_counts())

X = df[["jam_belajar", "kehadiran", "tugas"]]
y = df["lulus"]

X_train, X_test, y_train, y_test = train_test_split(
    X, y,
    test_size=0.3,
    random_state=42,
    stratify=y
)

scaler = StandardScaler()
X_train_scaled = scaler.fit_transform(X_train)
X_test_scaled = scaler.transform(X_test)

model = LogisticRegression()
model.fit(X_train_scaled, y_train)

y_pred = model.predict(X_test_scaled)

print("\nAccuracy:", accuracy_score(y_test, y_pred))
print("\nConfusion Matrix:")
print(confusion_matrix(y_test, y_pred))
print("\nClassification Report:")
print(classification_report(y_test, y_pred))
```

Pelajaran dari contoh ini

Pada data tidak seimbang, pembaca harus lebih memperhatikan:

- precision
- recall
- F1-score
- confusion matrix

daripada puas hanya dengan accuracy. Itu sangat ditekankan oleh materi lampiran.

22.12 Source code Python lengkap: evaluasi klasifikasi terpadu

Berikut satu program lengkap yang menggabungkan **confusion matrix, accuracy, precision, recall, F1-score, ROC-AUC, dan class imbalance dasar**.

Simpan sebagai:

```
src/bab22_evaluasi_klasifikasi_terpadu.py
```

```
import pandas as pd
import matplotlib.pyplot as plt
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import (
    confusion_matrix,
    accuracy_score,
    precision_score,
    recall_score,
    f1_score,
    classification_report,
    roc_curve,
    roc_auc_score
)

print("=" * 80)
print("BAB 22 - EVALUASI MODEL KLASIFIKASI")
print("=" * 80)

# 1. Load data
df = pd.read_csv("data/raw/evaluasi_klasifikasi_mahasiswa.csv")

print("\n[1] DATA AWAL")
print(df.head())

print("\nDistribusi kelas:")
print(df["lulus"].value_counts())

# 2. Fitur dan target
X = df[["jam_belajar", "kehadiran", "tugas"]]
y = df["lulus"]

# 3. Split data
X_train, X_test, y_train, y_test = train_test_split(
    X, y,
    test_size=0.25,
    random_state=42,
    stratify=y
)

# 4. Scaling
scaler = StandardScaler()
X_train_scaled = scaler.fit_transform(X_train)
X_test_scaled = scaler.transform(X_test)

# 5. Model
model = LogisticRegression()
model.fit(X_train_scaled, y_train)
```

```
# 6. Prediksi
y_pred = model.predict(X_test_scaled)
y_prob = model.predict_proba(X_test_scaled)[:, 1]

# 7. Metrik dasar
cm = confusion_matrix(y_test, y_pred)
acc = accuracy_score(y_test, y_pred)
precision = precision_score(y_test, y_pred)
recall = recall_score(y_test, y_pred)
f1 = f1_score(y_test, y_pred)
auc = roc_auc_score(y_test, y_prob)

print("\n[2] CONFUSION MATRIX")
print(cm)

print("\n[3] METRIK DASAR")
print("Accuracy :", acc)
print("Precision:", precision)
print("Recall   :", recall)
print("F1-score :", f1)
print("ROC-AUC  :", auc)

print("\n[4] CLASSIFICATION REPORT")
print(classification_report(y_test, y_pred))

# 8. ROC curve
fpr, tpr, thresholds = roc_curve(y_test, y_prob)

plt.figure(figsize=(7, 5))
plt.plot(fpr, tpr)
plt.plot([0, 1], [0, 1], linestyle="--")
plt.xlabel("False Positive Rate")
plt.ylabel("True Positive Rate")
plt.title("ROC Curve")
plt.grid(True)
plt.tight_layout()
plt.show()

print("\nSELESAI. Anda sudah mengevaluasi model klasifikasi secara lebih
lengkap.")
print("=" * 80)
```

22.13 Cara membaca hasil evaluasi dengan sehat

Agar pembaca tidak terjebak pada satu angka, berikut cara berpikir yang saya sarankan.

- **Confusion matrix**
  lihat dulu distribusi benar-salah per kelas
- **Accuracy**
  bagus untuk gambaran umum, tetapi jangan berdiri sendiri
- **Precision**
  penting bila *false positive* mahal
- **Recall**
  penting bila *false negative* mahal
- **F1-score**
  bagus untuk keseimbangan precision–recall, terutama saat data tidak seimbang
- **ROC-AUC**
  bagus untuk melihat kualitas diskriminasi di berbagai threshold, tetapi tetap harus dibaca bersama konteks imbalance
- **Distribusi kelas**
  selalu cek sebelum menafsirkan metrik

Materi lampiran juga sangat konsisten bahwa evaluasi klasifikasi harus dibaca secara **kontekstual dan multi-metrik**, bukan satu angka tunggal.

22.14 Kesalahan umum pemula

Ada beberapa jebakan yang sangat sering terjadi.

- **Hanya melihat accuracy**
  padahal data bisa sangat tidak seimbang.
- **Tidak membaca confusion matrix**
  sehingga tidak tahu jenis kesalahan model.
- **Mengira precision tinggi berarti model sudah bagus**
  padahal recall bisa sangat rendah.
- **Mengira recall tinggi pasti cukup**
  padahal precision bisa buruk dan membuat banyak alarm palsu.
- **Tidak memeriksa class imbalance**
  lalu merasa accuracy tinggi berarti model aman.
- **Membaca ROC-AUC tanpa konteks**
  terutama pada data sangat tidak seimbang.

22.15 Refleksi: evaluasi klasifikasi adalah soal memahami konsekuensi kesalahan

Bab ini seharusnya membuat pembaca menyadari bahwa evaluasi klasifikasi bukan sekadar urusan angka, tetapi juga **urusan jenis kesalahan yang paling berbahaya**. Pada masalah tertentu, *false positive* lebih mahal. Pada masalah lain, *false negative* jauh lebih berbahaya. Itulah sebabnya precision, recall, dan F1-score sangat penting, dan mengapa confusion matrix harus dibaca secara serius. Van Der Post menekankan hal ini sangat jelas melalui contoh medis dan fraud detection, sementara Knox menunjukkan bahwa ROC juga membantu mendiskusikan *operating point* yang sesuai dengan kebutuhan klien atau konteks keputusan.

Dengan kata lain, evaluasi klasifikasi mengajarkan pembaca satu pelajaran besar:
model yang baik bukan yang sekadar sering benar, tetapi yang salahnya juga bisa dipahami dan diterima dalam konteks masalah.

Penutup Bab

Bab ini membahas **evaluasi model klasifikasi** melalui **confusion matrix, accuracy, precision, recall, F1-score, ROC-AUC secara intuitif, dan class imbalance dasar**. Seluruh pembahasan ini sangat relevan dengan materi lampiran yang menekankan bahwa accuracy saja sering tidak cukup, confusion matrix memberi gambaran lebih kaya, precision–recall–F1 penting untuk membaca kualitas prediksi positif, ROC-AUC membantu melihat performa di berbagai threshold, dan class imbalance bisa membuat evaluasi menjadi menyesatkan bila dibaca secara dangkal.

Pesan terpenting dari bab ini: model klasifikasi yang baik bukan hanya yang akurasinya tinggi, tetapi yang **dievaluasi dengan metrik yang sesuai dengan risiko, distribusi kelas, dan tujuan nyata dari masalah yang sedang diselesaikan**.

Tantangan kecil untuk pembaca: modifikasi file `bab22_evaluasi_klasifikasi_terpadu.py` agar selain `LogisticRegression`, Anda juga membandingkan hasil `DecisionTreeClassifier` dan `RandomForestClassifier`, lalu lihat apakah model dengan ROC-AUC tertinggi juga selalu punya precision, recall, dan F1-score terbaik.

Bab 23. Unsupervised Learning I — Clustering

Bab ini memperkenalkan satu keluarga penting dalam *machine learning* yang berbeda dari regresi dan klasifikasi: **clustering**. Jika pada *supervised learning* kita punya label target yang jelas, maka pada *clustering* kita bekerja dengan data **tanpa label**, lalu mencoba menemukan pola kelompok alami yang tersembunyi di dalamnya. Materi lampiran sangat sejalan dengan arah ini: *clustering* dijelaskan sebagai teknik *unsupervised learning* yang mengelompokkan data berdasarkan kemiripan, dengan tujuan agar anggota dalam satu cluster lebih mirip satu sama lain dibanding anggota di cluster lain. K-Means juga disebut sebagai algoritma yang paling populer dan intuitif untuk memulai pembelajaran clustering, terutama pada kasus seperti **segmentasi pelanggan**.

Bab ini sangat penting karena membantu pembaca memahami bahwa tidak semua masalah AI membutuhkan label. Banyak kasus nyata justru dimulai dari pertanyaan seperti: “pelanggan saya sebenarnya terbagi menjadi kelompok seperti apa?”, “apakah ada pola perilaku belanja yang berbeda?”, atau “apakah data ini membentuk segmen alami?” Itulah wilayah clustering. Materi lampiran juga menekankan bahwa K-Means sangat berguna untuk **customer segmentation**, terutama ketika kita ingin mengelompokkan pelanggan berdasarkan fitur seperti pendapatan, frekuensi transaksi, atau *spending score*.

Bab ini dirancang agar pembaca mampu:

- memahami **apa itu clustering**
- menjalankan **K-Means**
- memilih **jumlah cluster**
- membuat **visualisasi cluster**
- memahami **studi kasus segmentasi pelanggan**
- menjalankan **contoh source code Python** yang langsung bisa dicoba di Ubuntu 26.04

23.1 Apa itu clustering

Clustering adalah teknik *unsupervised learning* untuk mengelompokkan data menjadi beberapa grup berdasarkan kemiripan. Berbeda dari klasifikasi, kita **tidak punya label jawaban** di awal. Model mencoba menemukan struktur laten atau pola pengelompokan dari data itu sendiri. Materi lampiran menjelaskan bahwa tujuan utamanya adalah membuat data dalam satu cluster memiliki kemiripan tinggi, sementara data antar cluster cukup berbeda. Ini sangat relevan untuk bisnis, pemasaran, bioinformatika, dan eksplorasi data pada umumnya.

Intuisi sederhana

Bayangkan Anda punya data pelanggan dengan dua kolom:

- pendapatan tahunan
- skor belanja

Anda belum tahu label “tipe pelanggan”. Namun setelah dilihat, ternyata ada kelompok:

- pelanggan pendapatan rendah, belanja rendah
- pelanggan pendapatan tinggi, belanja tinggi
- pelanggan pendapatan tinggi, belanja rendah

Clustering membantu menemukan kelompok semacam itu **tanpa label manual**. Lampiran juga memberi contoh yang sangat mirip pada segmentasi pelanggan di dunia ritel dan finansial.

Kapan clustering dipakai

Clustering cocok digunakan jika:

- tidak ada label target
- ingin melakukan segmentasi
- ingin eksplorasi pola awal
- ingin memahami struktur data sebelum modeling lanjutan

Namun materi lampiran juga mengingatkan agar tidak berlebihan menginterpretasikan hasil clustering, karena algoritma yang berbeda bisa memberi jawaban yang berbeda, dan clustering lebih bersifat heuristik dibanding supervised learning.

23.2 K-Means

Apa itu K-Means

K-Means adalah algoritma clustering berbasis centroid. Ia membagi data ke dalam K cluster, lalu setiap titik data ditempatkan ke centroid terdekat. Setelah itu centroid diperbarui sebagai rata-rata titik-titik dalam cluster, dan proses ini diulang sampai hasil stabil. Materi lampiran menjelaskan dua langkah inti K-Means dengan sangat jelas: pertama, setiap titik diassign ke pusat cluster terdekat; kedua, pusat cluster diperbarui menjadi mean dari anggota cluster tersebut. K-Means juga sangat terkait dengan **jarak Euclidean kuadrat**, sehingga ia paling cocok pada data numerik dengan cluster yang cukup kompak dan “membulat”.

Kenapa K-Means populer

K-Means populer karena:

- relatif mudah dipahami
- cepat dijalankan
- mudah divisualisasikan
- sangat cocok untuk pengantar clustering
- umum dipakai untuk segmentasi pelanggan

Materi lampiran juga menegaskan bahwa K-Means adalah salah satu algoritma clustering yang paling sederhana dan paling banyak digunakan.

Catatan penting tentang K-Means

K-Means punya beberapa keterbatasan penting:

- sensitif terhadap **pemilihan centroid awal**
- sensitif terhadap **outlier**
- kurang cocok untuk cluster dengan bentuk sangat tidak beraturan
- butuh kita menentukan **jumlah cluster K** sejak awal

Lampiran Uddin menekankan bahwa centroid awal yang buruk bisa membawa K-Means ke solusi lokal yang kurang optimal, dan bahwa algoritma ini tidak secara khusus menangani outlier.

23.3 Dataset contoh untuk bab ini

Agar pembahasan lebih praktis, kita akan memakai dataset sederhana bertema **segmentasi pelanggan**.

Simpan sebagai:

```
data/raw/segmentasi_pelanggan.csv
```

Isi file:

```
pelanggan,pendapatan_tahunan,skor_belanja
P1,15,39
P2,16,81
P3,17,6
P4,18,77
P5,35,40
P6,36,76
P7,37,6
P8,38,72
P9,60,55
P10,62,52
P11,64,59
P12,66,54
```

```
P13,90,20
P14,92,18
P15,94,22
P16,96,19
```

Kolom yang akan kita pakai untuk clustering:

- `pendapatan_tahunan`
- `skor_belanja`

23.4 Langkah awal: memuat dan melihat data

Seperti biasa, langkah pertama adalah membaca data dan melihat bentuk awalnya.

Simpan sebagai:

```
src/bab23_load_data_clustering.py

import pandas as pd

df = pd.read_csv("data/raw/segmentasi_pelanggan.csv")

print("=== DATA AWAL ===")
print(df)

print("\n=== DESCRIBE ===")
print(df.describe())
```

Langkah ini penting karena lampiran juga menekankan bahwa sebelum clustering, kita perlu memahami skala dan distribusi fitur yang akan dipakai. Pada studi kasus segmentasi pelanggan, langkah eksplorasi awal diposisikan sebagai tahap fondasional sebelum normalisasi dan clustering.

23.5 Mengapa scaling penting sebelum K-Means

Karena K-Means menggunakan jarak, fitur yang skalanya jauh lebih besar bisa mendominasi hasil cluster. Itulah sebabnya materi lampiran pada studi kasus segmentasi pelanggan secara eksplisit menekankan **normalization** atau **standardization** sebelum K-Means, agar setiap fitur memberi kontribusi yang lebih seimbang terhadap perhitungan jarak.

Source code Python lengkap

Simpan sebagai:

```
src/bab23_scaling_kmeans.py

import pandas as pd
from sklearn.preprocessing import StandardScaler

df = pd.read_csv("data/raw/segmentasi_pelanggan.csv")

X = df[["pendapatan_tahunan", "skor_belanja"]]

scaler = StandardScaler()
X_scaled = scaler.fit_transform(X)

print("=== DATA SETELAH SCALING ===")
print(X_scaled[:5])
```

Intuisi

Kalau satu fitur berada di rentang 0–100 dan fitur lain 0–1.000.000, maka tanpa scaling model akan lebih “mendengar” fitur besar. Karena itu, scaling sering menjadi kebiasaan sehat pada clustering berbasis jarak.

23.6 Menjalankan K-Means

Sekarang kita mulai clustering inti.

Simpan sebagai:

```
src/bab23_kmeans_dasar.py

import pandas as pd
from sklearn.cluster import KMeans
from sklearn.preprocessing import StandardScaler

# 1. Load data
df = pd.read_csv("data/raw/segmentasi_pelanggan.csv")

# 2. Pilih fitur
X = df[["pendapatan_tahunan", "skor_belanja"]]

# 3. Scaling
scaler = StandardScaler()
X_scaled = scaler.fit_transform(X)

# 4. K-Means
model = KMeans(n_clusters=3, random_state=42, n_init=10)
df["cluster"] = model.fit_predict(X_scaled)

print("=== HASIL CLUSTERING ===")
print(df)
```

```
print("\nCentroid (skala standar):")
print(model.cluster_centers_)
```

Apa yang terjadi di kode ini

- `n_clusters=3` berarti kita meminta 3 cluster
- `fit_predict()` melatih model dan langsung memberi label cluster
- hasil cluster ditulis ke kolom baru `cluster`

Ini sangat sejalan dengan contoh-contoh lampiran yang memakai `KMeans(...).fit(X)` lalu `predict(X)` atau `fit_predict(X)` untuk memberi label cluster ke data pelanggan.

23.7 Memilih jumlah cluster

Ini adalah pertanyaan paling penting dalam K-Means:

berapa jumlah cluster yang masuk akal?

Materi lampiran menekankan bahwa K-Means adalah algoritma *k-clustering*, artinya `k` adalah parameter yang ditentukan pengguna. Karena itu, kita perlu pendekatan praktis untuk memilih `k`. Selain intuisi domain, dua pendekatan yang sangat umum adalah:

- **Elbow Method**
- **Silhouette Score**

A. Elbow Method

Elbow Method melihat penurunan *within-cluster sum of squares* atau `inertia`. Ide sederhananya: tambahkan jumlah cluster sedikit demi sedikit, lalu cari titik “siku” saat penurunan tidak lagi terlalu besar.

Simpan sebagai:

```
src/bab23_elbow_method.py

import pandas as pd
import matplotlib.pyplot as plt
from sklearn.cluster import KMeans
from sklearn.preprocessing import StandardScaler

df = pd.read_csv("data/raw/segmentasi_pelanggan.csv")

X = df[["pendapatan_tahunan", "skor_belanja"]]

scaler = StandardScaler()
X_scaled = scaler.fit_transform(X)

inertia_values = []

for k in range(1, 8):
    model = KMeans(n_clusters=k, random_state=42, n_init=10)
    model.fit(X_scaled)
    inertia_values.append(model.inertia_)

plt.figure(figsize=(8, 5))
plt.plot(range(1, 8), inertia_values, marker="o")
plt.title("Elbow Method")
plt.xlabel("Jumlah Cluster (k)")
plt.ylabel("Inertia")
plt.grid(True)
plt.tight_layout()
plt.show()
```

Cara membaca Elbow Method

Cari titik saat penurunan inertia mulai melambat tajam. Titik itu sering menjadi kandidat k yang masuk akal.

B. Silhouette Score

Lampiran menjelaskan **Silhouette Score** sebagai metrik intrinsik yang menggabungkan dua hal penting:

- **cohesion** → seberapa rapat anggota dalam satu cluster
- **separation** → seberapa jelas pemisahan antar cluster

Nilainya berada di rentang -1 sampai 1:

- mendekati **+1** → cluster cukup jelas
- sekitar **0** → banyak tumpang tindih
- negatif → cluster buruk

Simpan sebagai:

```
src/bab23_silhouette_score.py

import pandas as pd
from sklearn.cluster import KMeans
from sklearn.metrics import silhouette_score
from sklearn.preprocessing import StandardScaler

df = pd.read_csv("data/raw/segmentasi_pelanggan.csv")

X = df[["pendapatan_tahunan", "skor_belanja"]]

scaler = StandardScaler()
X_scaled = scaler.fit_transform(X)

for k in range(2, 7):
    model = KMeans(n_clusters=k, random_state=42, n_init=10)
    labels = model.fit_predict(X_scaled)
    score = silhouette_score(X_scaled, labels)
    print(f"k={k}, silhouette_score={score:.4f}")
```

Cara memakai dua pendekatan ini bersama

Praktik yang sehat:

- lihat **Elbow Method**
- cek **Silhouette Score**
- cocokkan dengan **intuisi bisnis atau domain**

Tidak ada satu aturan absolut, tetapi kombinasi ini sangat membantu.

23.8 Visualisasi cluster

Visualisasi adalah bagian yang sangat penting dalam clustering, karena tanpa label target, kita sangat mengandalkan visual untuk membaca struktur kelompok. Materi lampiran juga berkali-kali menampilkan visualisasi cluster pelanggan sebagai cara utama untuk memahami hasil segmentasi.

Source code Python lengkap

Simpan sebagai:

```
src/bab23_visualisasi_cluster.py

import pandas as pd
import matplotlib.pyplot as plt
from sklearn.cluster import KMeans
from sklearn.preprocessing import StandardScaler

# 1. Load data
df = pd.read_csv("data/raw/segmentasi_pelanggan.csv")

# 2. Pilih fitur
X = df[["pendapatan_tahunan", "skor_belanja"]]

# 3. Scaling
scaler = StandardScaler()
X_scaled = scaler.fit_transform(X)

# 4. K-Means
model = KMeans(n_clusters=3, random_state=42, n_init=10)
df["cluster"] = model.fit_predict(X_scaled)

# 5. Visualisasi
plt.figure(figsize=(8, 5))
plt.scatter(df["pendapatan_tahunan"], df["skor_belanja"], c=df["cluster"])
plt.xlabel("Pendapatan Tahunan")
plt.ylabel("Skor Belanja")
plt.title("Visualisasi Cluster Pelanggan")
plt.grid(True)
plt.tight_layout()
plt.show()
```

Apa yang harus diperhatikan

Saat melihat grafik cluster, perhatikan:

- apakah kelompok terlihat cukup terpisah
- apakah ada titik yang sangat terpencil
- apakah cluster tampak masuk akal secara bisnis
- apakah jumlah cluster terlalu sedikit atau terlalu banyak

Lampiran juga mengingatkan bahwa visualisasi sering lebih efektif daripada sekadar angka saat ingin menemukan pengelompokan yang menarik, tetapi kita tetap harus berhati-hati agar tidak **over-interpretation**.

23.9 Studi kasus segmentasi pelanggan

Sekarang kita satukan semuanya ke dalam satu mini studi kasus.

Masalah

Sebuah toko ingin memahami pola pelanggan berdasarkan:

- **pendapatan tahunan**
- **skor belanja**

Mereka belum punya label resmi seperti “pelanggan hemat”, “pelanggan premium”, atau “pelanggan impulsif”. Maka mereka memakai clustering untuk menemukan kelompok alami lebih dulu.

Materi lampiran memberi studi kasus yang sangat mirip: perusahaan ritel atau bank ingin mengelompokkan pelanggan berdasarkan perilaku transaksi, *spending score*, atau aktivitas akun untuk mendukung kampanye yang lebih tertarget dan layanan yang lebih personal.

Alur kerja studi kasus

- load data pelanggan
- pilih fitur yang relevan
- lakukan scaling
- tentukan jumlah cluster
- jalankan K-Means
- visualisasikan cluster
- baca profil tiap cluster
- tulis insight awal

Source code Python lengkap: studi kasus terpadu

Simpan sebagai:

src/bab23_studi_kasus_segmentasi_pelanggan.py

```
import pandas as pd
import matplotlib.pyplot as plt
from sklearn.cluster import KMeans
from sklearn.metrics import silhouette_score
from sklearn.preprocessing import StandardScaler

print("=" * 80)
print("BAB 23 - UNSUPERVISED LEARNING I: CLUSTERING")
print("=" * 80)

# 1. Load data
df = pd.read_csv("data/raw/segmentasi_pelanggan.csv")

print("\n[1] DATA AWAL")
print(df)

# 2. Pilih fitur
X = df[["pendapatan_tahunan", "skor_belanja"]]

# 3. Scaling
scaler = StandardScaler()
X_scaled = scaler.fit_transform(X)

# 4. Coba beberapa nilai k
print("\n[2] SILHOUETTE SCORE UNTUK BEBERAPA K")
for k in range(2, 6):
    model = KMeans(n_clusters=k, random_state=42, n_init=10)
    labels = model.fit_predict(X_scaled)
    score = silhouette_score(X_scaled, labels)
    print(f"k={k}, silhouette_score={score:.4f}")

# 5. Final model
k_final = 3
model = KMeans(n_clusters=k_final, random_state=42, n_init=10)
df["cluster"] = model.fit_predict(X_scaled)

print("\n[3] HASIL CLUSTER")
print(df)

# 6. Ringkasan cluster
ringkasan = df.groupby("cluster")[["pendapatan_tahunan",
"skor_belanja"]].mean()
print("\n[4] RATA-RATA TIAP CLUSTER")
print(ringkasan)

# 7. Visualisasi
plt.figure(figsize=(8, 5))
plt.scatter(df["pendapatan_tahunan"], df["skor_belanja"], c=df["cluster"])
plt.xlabel("Pendapatan Tahunan")
plt.ylabel("Skor Belanja")
plt.title("Segmentasi Pelanggan dengan K-Means")
plt.grid(True)
plt.tight_layout()
```

```
plt.show()

print("\n[5] INSIGHT AWAL")
print("- Cluster yang berbeda menunjukkan pola pelanggan yang berbeda.")
print("- Ada kelompok pelanggan dengan pendapatan tinggi tetapi skor belanja
rendah.")
print("- Ada juga kelompok pelanggan dengan pendapatan menengah/tinggi dan
skor belanja lebih aktif.")
print("- Hasil ini bisa dipakai sebagai dasar strategi pemasaran yang lebih
terarah.")

print("\nSELESAI. Anda sudah menjalankan studi kasus segmentasi pelanggan.")
print("=" * 80)
```

23.10 Cara membaca hasil segmentasi

Setelah cluster terbentuk, pertanyaan berikutnya bukan lagi “berapa label cluster-nya”, tetapi:

apa arti tiap cluster itu secara bisnis atau konteks?

Misalnya:

- **Cluster 0** → pendapatan rendah, belanja rendah
- **Cluster 1** → pendapatan tinggi, belanja tinggi
- **Cluster 2** → pendapatan tinggi, belanja rendah

Dari sini toko bisa membuat strategi berbeda:

- promosi hemat untuk cluster sensitif harga
- produk premium untuk cluster belanja tinggi
- kampanye aktivasi ulang untuk pendapatan tinggi tetapi belanja rendah

Materi lampiran juga menekankan bahwa nilai clustering muncul saat hasilnya diterjemahkan menjadi strategi yang dapat ditindaklanjuti, misalnya kampanye pemasaran yang lebih personal atau layanan yang lebih sesuai tiap segmen.

23.11 Kelebihan dan keterbatasan K-Means

Kelebihan K-Means

- mudah dipahami
- cepat
- cocok untuk pengantar clustering
- sangat baik untuk segmentasi awal
- mudah divisualisasikan

Keterbatasan K-Means

- harus menentukan k di awal
- sensitif terhadap outlier
- sensitif terhadap centroid awal
- paling cocok untuk cluster yang cukup bulat/kompak
- tidak otomatis mendeteksi outlier

Lampiran Uddin menegaskan bahwa K-Means **tidak menekankan identifikasi outlier**, berbeda dengan DBSCAN atau HDBSCAN. Knox juga menjelaskan bahwa K-Means terkait erat dengan centroid mean dan jarak Euclidean, sehingga bentuk cluster sangat mempengaruhi hasil.

Kapan perlu mempertimbangkan algoritma lain

Kalau data:

- penuh outlier
- cluster-nya tidak bulat
- kepadatan cluster sangat berbeda
- bentuk cluster rumit

maka algoritma lain seperti **DBSCAN** atau **hierarchical clustering** bisa jadi lebih cocok. Lampiran Van Der Post dan Uddin menyinggung hal ini dengan cukup jelas.

23.12 Kesalahan umum pemula

Ada beberapa jebakan yang sangat sering terjadi.

- **langsung menjalankan K-Means tanpa scaling**
  padahal fitur bisa punya skala sangat berbeda.
- **menganggap hasil cluster sebagai kebenaran mutlak**
  padahal clustering bersifat heuristik dan algoritma berbeda bisa memberi hasil berbeda.
- **memilih jumlah cluster asal-asalan**
  padahal k harus dipertimbangkan lewat elbow, silhouette, dan konteks domain.
- **mengabaikan outlier**
  padahal K-Means sensitif terhadap outlier dan centroid awal.
- **hanya melihat plot tanpa menulis insight**
  padahal tujuan akhirnya adalah segmentasi yang bisa diinterpretasikan dan dipakai.

23.13 Refleksi: clustering adalah alat eksplorasi, bukan mesin kebenaran

Bab ini seharusnya membuat pembaca melihat bahwa clustering sangat berguna, tetapi juga harus dipakai dengan sikap hati-hati. Ia membantu kita menemukan **struktur awal**, **segmen pelanggan**, atau **kelompok pola** yang sebelumnya tidak terlihat. Namun hasilnya tetap perlu dibaca bersama:

- visualisasi
- ringkasan statistik tiap cluster
- konteks bisnis
- pengetahuan domain

Materi lampiran menekankan hal ini dari dua sisi: satu sisi, clustering sangat kuat untuk *customer segmentation* dan eksplorasi pola; sisi lain, ada resiko *over-interpretation* dan hasil antara algoritma bisa berbeda. Jadi clustering paling baik dipakai sebagai **alat eksplorasi yang cerdas**, bukan penentu kebenaran final.

Penutup Bab

Bab ini membahas **Unsupervised Learning I — Clustering** melalui pengenalan **apa itu clustering**, implementasi **K-Means**, cara **memilih jumlah cluster**, membuat **visualisasi cluster**, serta **studi kasus segmentasi pelanggan** dengan **contoh source code Python lengkap**. Seluruh pembahasan ini sangat relevan dengan materi lampiran yang menempatkan clustering sebagai teknik *unsupervised learning* untuk menemukan kelompok alami pada data tanpa label, dengan **K-Means** sebagai algoritma yang paling intuitif dan paling populer untuk pengantar segmentasi pelanggan.

Pesan terpenting dari bab ini: clustering bukan sekadar memberi nomor cluster, tetapi membantu kita **melihat pola tersembunyi, menyusun segmentasi yang lebih bermakna, dan membuka pertanyaan analitis baru**.

Tantangan kecil untuk pembaca: modifikasi file `bab23_studi_kasus_segmentasi_pelanggan.py` agar selain `k=3`, Anda juga mencoba `k=2`, `k=4`, dan `k=5`, lalu bandingkan **silhouette score**, visualisasi cluster, dan interpretasi bisnisnya.

Bab 24. Unsupervised Learning II — Reduksi Dimensi

Bab ini melanjutkan jalur *unsupervised learning* dari clustering ke topik yang sangat penting untuk analisis data modern, yaitu **reduksi dimensi**. Di outline buku utama, fokus bab ini memang pada **PCA secara intuitif**, **mengapa dimensi perlu dikurangi**, **visualisasi 2D dari data berdimensi lebih tinggi**, dan **contooh source code Python**. Arah ini sangat selaras dengan materi lampiran yang membahas **supervised vs unsupervised learning**, **K-Means**, **PCA**, **model evaluation**, dan peran PCA sebagai alat untuk menyederhanakan data yang kompleks agar lebih mudah divisualisasikan dan dianalisis.

Materi lampiran juga cukup konsisten dalam menjelaskan peran PCA. Van Der Post menegaskan bahwa **Principal Component Analysis (PCA)** adalah teknik reduksi dimensi yang mentransformasikan data menjadi sekumpulan komponen ortogonal yang menangkap **variasi terbesar** dalam data, sehingga visualisasi dan analisis menjadi lebih mudah. Referensi lain menambahkan bahwa pada data berdimensi tinggi, PCA juga dapat membantu preprocessing sebelum tugas lain seperti clustering atau klasifikasi. Bahkan pada konteks KNN, lampiran menyebut bahwa reduksi dimensi dapat membantu mengatasi **curse of dimensionality**, yaitu melemahnya efektivitas perhitungan jarak saat jumlah fitur terlalu banyak.

Bab ini dirancang agar pembaca mampu:

- memahami **mengapa dimensi perlu dikurangi**
- memahami **PCA secara intuitif**
- membuat **visualisasi 2D** dari data yang semula berdimensi lebih tinggi
- menjalankan **contoh source code Python**
- memahami kapan PCA berguna dan apa keterbatasannya

24.1 Mengapa dimensi perlu dikurangi

Saat jumlah fitur bertambah, data bisa menjadi sulit dibaca, sulit divisualisasikan, dan kadang lebih sulit diproses model. Lampiran menjelaskan bahwa daya tarik machine learning justru sering muncul saat ia menangani **high-dimensional data**, tetapi kompleksitas itu juga membawa tantangan. PCA hadir sebagai salah satu cara untuk menyederhanakan data tanpa langsung membuang struktur penting yang terkandung di dalamnya.

Secara praktis, reduksi dimensi dibutuhkan karena beberapa alasan berikut.

- **Visualisasi manusia terbatas.** Kita mudah melihat data 2D atau 3D, tetapi sulit memahami 10, 50, atau 500 fitur sekaligus. Lampiran menyebut bahwa PCA sangat berguna untuk membuat visualisasi data berdimensi tinggi menjadi lebih mudah dibaca.
- **Banyak fitur bisa saling tumpang tindih informasi.** Jika beberapa fitur membawa informasi yang mirip, PCA dapat merangkum informasi itu ke komponen yang lebih sedikit namun tetap kuat. Lampiran Lim menjelaskan bahwa komponen utama pertama dan seterusnya dibentuk untuk menjelaskan proporsi varians terbesar dalam data.

- **Performa beberapa algoritma bisa terganggu oleh dimensi tinggi.** Pada KNN misalnya, lampiran Van Der Post menjelaskan bahwa perhitungan jarak menjadi kurang efektif saat dimensi terlalu tinggi, fenomena yang dikenal sebagai *curse of dimensionality*, dan PCA bisa membantu mereduksinya.
- **Preprocessing untuk tugas lanjutan.** Lampiran juga menekankan bahwa PCA bisa membantu sebelum clustering atau klasifikasi, karena data menjadi lebih ringkas dan lebih mudah dipelajari model.

Dengan kata lain, reduksi dimensi bukan sekadar “membuang kolom”, tetapi **merangkum informasi penting** agar data menjadi lebih mudah dipahami dan lebih efektif dipakai.

24.2 Apa itu PCA secara intuitif

PCA (*Principal Component Analysis*) adalah teknik yang mengubah sekumpulan fitur asli menjadi sekumpulan fitur baru yang disebut **principal components**. Komponen-komponen ini disusun agar:

- saling tidak berkorelasi
- menangkap variasi terbesar dalam data
- diurutkan dari yang paling banyak menjelaskan variasi ke yang lebih kecil

Lampiran Van Der Post menjelaskan bahwa PCA menyederhanakan dataset kompleks dengan mencari dimensi yang paling signifikan terhadap variasi data, sehingga visualisasi dan analisis menjadi lebih mudah. Referensi lain menambahkan bahwa PCA mentransformasikan data ke komponen ortogonal atau tidak berkorelasi.

Intuisi sederhana

Bayangkan Anda punya data 2D, tetapi titik-titiknya membentuk awan data miring diagonal. Kalau kita tetap melihatnya pada sumbu X dan Y asli, variasinya terasa “tersebar”. PCA mencoba mencari **arah baru** yang paling sesuai dengan sebaran utama data itu. Arah pertama itulah yang disebut **principal component 1**, dan arah kedua yang tegak lurus terhadapnya menjadi **principal component 2**. Lampiran Lim menjelaskan hal ini secara lebih matematis: setiap komponen utama adalah kombinasi linear dari fitur asli, dan varians komponen itu sama dengan eigenvalue yang terkait dengannya.

Apa yang dipertahankan PCA

PCA berusaha mempertahankan **variasi terbesar** dalam data. Jadi kalau kita mengurangi dari 10 dimensi menjadi 2, yang kita simpan adalah dua arah yang paling banyak menjelaskan struktur variasi data. Itulah sebabnya PCA sering dipakai untuk membuat ringkasan dimensi yang masih “bermakna”. Lampiran Van Der Post menekankan bahwa komponen dengan eigenvalue terbesar akan menangkap variasi paling signifikan.

24.3 PCA bukan sekadar memilih kolom

Penting dipahami bahwa PCA **bukan** sekadar memilih dua kolom terbaik. PCA membuat **kombinasi baru** dari fitur asli. Itu sebabnya hasil PCA biasanya tidak lagi punya nama seperti `umur`, `pendapatan`, atau `skor_belanja`, tetapi menjadi:

- `PC1`
- `PC2`
- `PC3`, dan seterusnya

Lampiran menjelaskan bahwa principal components adalah kombinasi linear dari variabel asli. Ini juga berarti PCA kadang membuat interpretasi menjadi sedikit lebih sulit, karena komponen baru tidak selalu punya makna langsung yang sesederhana fitur asli. Van Der Post secara eksplisit menyebut bahwa salah satu keterbatasan PCA adalah komponen hasilnya kadang sulit diberi makna yang sangat jelas.

Jadi, **keuntungan PCA adalah penyederhanaan**, tetapi **konsekuensinya kadang pada interpretasi**.

24.4 Dataset contoh untuk bab ini

Agar pembahasan tetap *hands-on*, kita akan memakai dataset sintetis berdimensi lebih tinggi. Ini cocok dengan semangat buku dan lampiran, karena beberapa referensi juga mencontohkan PCA dengan data sintetis dan menunjukkan langsung transformasi ke dua komponen utama.

Simpan file berikut sebagai:

```
data/raw/pca_data_sederhana.csv
```

Isi file:

```
f1,f2,f3,f4,f5
2.5,2.4,1.2,3.1,0.9
0.5,0.7,1.0,1.5,0.4
2.2,2.9,1.4,3.0,0.8
1.9,2.2,1.3,2.7,0.7
3.1,3.0,1.8,3.6,1.1
2.3,2.7,1.5,3.2,0.9
2.0,1.6,1.1,2.4,0.6
1.0,1.1,0.9,1.8,0.5
1.5,1.6,1.0,2.0,0.6
1.1,0.9,0.8,1.7,0.4
3.0,3.2,1.9,3.8,1.2
2.7,2.8,1.7,3.4,1.0
1.4,1.5,1.0,2.1,0.6
0.8,0.6,0.7,1.4,0.3
2.8,3.1,1.8,3.7,1.1
```

Dataset ini punya 5 fitur numerik, cukup untuk menunjukkan bagaimana reduksi dimensi bekerja.

24.5 Langkah awal: memuat dan melihat data

Seperti biasa, kita mulai dari membaca data dan memeriksa bentuknya.

Simpan sebagai:

```
src/bab24_load_data_pca.py

import pandas as pd

df = pd.read_csv("data/raw/pca_data_sederhana.csv")

print("=== DATA AWAL ===")
print(df.head())

print("\nShape:", df.shape)
print("\nDeskripsi statistik:")
print(df.describe())
```

Tujuan langkah ini adalah memastikan pembaca benar-benar melihat bahwa kita memulai dari data dengan **lebih dari dua dimensi**, sehingga visualisasi langsung dengan scatter plot sederhana belum memadai.

24.6 Scaling sebelum PCA

Ini adalah langkah yang sangat penting. Lampiran Van Der Post secara eksplisit menyebut bahwa **standardizing the data is a crucial step in PCA**, karena PCA sensitif terhadap skala fitur. Jika satu fitur jauh lebih besar skalanya, maka komponen utama akan lebih banyak dipengaruhi oleh fitur itu, bukan oleh struktur variasi keseluruhan yang adil.

Mengapa scaling penting

> Bayangkan satu fitur berada di rentang 0–1 dan fitur lain di rentang 0–10.000. Tanpa scaling, PCA hampir pasti akan lebih “mendengar” fitur yang besar skalanya. Karena itu, sebelum PCA, biasanya kita melakukan standardisasi agar setiap fitur punya kontribusi yang lebih seimbang. Lampiran juga memberi contoh standardisasi data dengan mengurangi mean dan membagi standar deviasi sebelum menjalankan PCA.

Source code Python lengkap

Simpan sebagai:

```
src/bab24_scaling_sebelum_pca.py

import pandas as pd
from sklearn.preprocessing import StandardScaler

df = pd.read_csv("data/raw/pca_data_sederhana.csv")

X = df[["f1", "f2", "f3", "f4", "f5"]]

scaler = StandardScaler()
X_scaled = scaler.fit_transform(X)

print("=== DATA SETELAH SCALING ===")
print(X_scaled[:5])
```

24.7 Menjalankan PCA ke 2 dimensi

Sekarang kita masuk ke inti utama bab ini: mengubah data 5 dimensi menjadi 2 dimensi. Lampiran memberi contoh yang sangat serupa, yakni memakai `PCA(n_components=2)` lalu `fit_transform(X)` untuk menghasilkan data hasil reduksi yang siap divisualisasikan.

Source code Python lengkap

Simpan sebagai:

```
src/bab24_pca_2d.py

import pandas as pd
from sklearn.preprocessing import StandardScaler
from sklearn.decomposition import PCA

# 1. Load data
df = pd.read_csv("data/raw/pca_data_sederhana.csv")

# 2. Ambil fitur
X = df[["f1", "f2", "f3", "f4", "f5"]]

# 3. Scaling
scaler = StandardScaler()
X_scaled = scaler.fit_transform(X)

# 4. PCA ke 2 dimensi
pca = PCA(n_components=2)
X_pca = pca.fit_transform(X_scaled)

# 5. Ubah ke DataFrame
df_pca = pd.DataFrame(X_pca, columns=["PC1", "PC2"])
```

```
print("=== HASIL PCA 2D ===")
print(df_pca.head())

print("\nExplained variance ratio:")
print(pca.explained_variance_ratio_)
```

Apa itu explained_variance_ratio_

Nilai ini menunjukkan berapa proporsi variasi data yang dijelaskan oleh setiap komponen utama. Misalnya kalau hasilnya:

- `PC1 = 0.72`
- `PC2 = 0.18`

maka dua komponen pertama bersama-sama menjelaskan sekitar **90% variasi data**. Lampiran menjelaskan bahwa komponen terbesar menjelaskan varians terbesar, sehingga proporsi ini penting untuk menilai apakah reduksi dimensi masih cukup representatif.

24.8 Visualisasi 2D dari data berdimensi lebih tinggi

Ini adalah salah satu manfaat paling langsung dari PCA: data yang awalnya sulit divisualisasikan kini bisa dipetakan ke scatter plot 2D. Lampiran menekankan bahwa PCA membuat data berdimensi tinggi menjadi lebih mudah divisualisasikan dan dapat membantu menemukan hubungan, cluster, atau outlier yang tadinya sulit dilihat.

Source code Python lengkap

Simpan sebagai:

```
src/bab24_visualisasi_pca_2d.py

import pandas as pd
import matplotlib.pyplot as plt
from sklearn.preprocessing import StandardScaler
from sklearn.decomposition import PCA

# 1. Load data
df = pd.read_csv("data/raw/pca_data_sederhana.csv")

# 2. Ambil fitur
X = df[["f1", "f2", "f3", "f4", "f5"]]

# 3. Scaling
scaler = StandardScaler()
X_scaled = scaler.fit_transform(X)

# 4. PCA
pca = PCA(n_components=2)
X_pca = pca.fit_transform(X_scaled)
```

```
# 5. DataFrame hasil
df_pca = pd.DataFrame(X_pca, columns=["PC1", "PC2"])

# 6. Visualisasi
plt.figure(figsize=(8, 5))
plt.scatter(df_pca["PC1"], df_pca["PC2"])
plt.title("Visualisasi 2D dengan PCA")
plt.xlabel("Principal Component 1")
plt.ylabel("Principal Component 2")
plt.grid(True)
plt.tight_layout()
plt.show()
```

Apa yang bisa dibaca dari visualisasi ini

Dari scatter plot hasil PCA, pembaca bisa mulai melihat:

- apakah ada kelompok titik yang mirip
- apakah ada outlier
- apakah data menyebar memanjang di arah tertentu
- apakah ada kemungkinan cluster yang sebelumnya tersembunyi

Lampiran Van Der Post juga menekankan hal ini secara eksplisit: scatter plot PCA membantu menyingkap hubungan, cluster, atau outlier yang tersembunyi dalam dimensi yang lebih tinggi.

24.9 Studi kecil: PCA sebelum clustering

Salah satu hubungan paling menarik antara bab ini dan bab sebelumnya adalah bahwa PCA sering sangat membantu **sebelum clustering**. Lampiran menyebut bahwa PCA dapat meningkatkan preprocessing untuk tugas seperti clustering atau classification. Pada studi segmentasi pelanggan, PCA juga bisa dipakai untuk membantu visualisasi hasil cluster bila fitur awal banyak.

Contoh konsep

Bayangkan kita punya dataset pelanggan dengan 6–10 fitur. Langsung mem-plot semua fitur mustahil. Tetapi kita bisa:

1. scaling
2. PCA ke 2 dimensi
3. plot hasil PCA
4. memberi warna berdasarkan label cluster dari K-Means

Ini membuat hasil clustering lebih mudah dibaca secara visual.

Source code Python lengkap

Simpan sebagai:

```
src/bab24_pca_dan_kmeans.py

import pandas as pd
import matplotlib.pyplot as plt
from sklearn.preprocessing import StandardScaler
from sklearn.decomposition import PCA
from sklearn.cluster import KMeans

# 1. Load data
df = pd.read_csv("data/raw/pca_data_sederhana.csv")

# 2. Fitur
X = df[["f1", "f2", "f3", "f4", "f5"]]

# 3. Scaling
scaler = StandardScaler()
X_scaled = scaler.fit_transform(X)

# 4. KMeans untuk contoh segmentasi
kmeans = KMeans(n_clusters=3, random_state=42, n_init=10)
cluster_labels = kmeans.fit_predict(X_scaled)

# 5. PCA ke 2 dimensi
pca = PCA(n_components=2)
X_pca = pca.fit_transform(X_scaled)

df_pca = pd.DataFrame(X_pca, columns=["PC1", "PC2"])
df_pca["cluster"] = cluster_labels

# 6. Visualisasi
plt.figure(figsize=(8, 5))
plt.scatter(df_pca["PC1"], df_pca["PC2"], c=df_pca["cluster"])
plt.title("PCA 2D + Label Cluster")
plt.xlabel("PC1")
plt.ylabel("PC2")
plt.grid(True)
plt.tight_layout()
plt.show()

print("Explained variance ratio:", pca.explained_variance_ratio_)
```

Apa yang dipelajari dari contoh ini

Contoh ini menunjukkan bahwa PCA bukan hanya alat visualisasi, tetapi juga jembatan yang sangat baik antara:

- data berdimensi tinggi
- clustering
- eksplorasi pola
- komunikasi hasil

24.10 Kapan PCA sangat berguna

PCA sangat berguna jika:

- jumlah fitur cukup banyak
- ingin membuat visualisasi 2D atau 3D
- ingin menyederhanakan fitur sebelum analisis lanjut
- curiga ada redundansi informasi antar fitur
- ingin mengurangi efek dimensi tinggi pada algoritma tertentu

Lampiran Van Der Post secara eksplisit menyoroti bahwa PCA membantu interpretasi, menyoroti komponen dengan varians terbesar, dan dapat meningkatkan performa tugas lanjutan seperti clustering atau classification. Lampiran tentang KNN juga menyebut PCA sebagai alat untuk mengurangi dampak *curse of dimensionality*.

24.11 Keterbatasan PCA

Walaupun sangat berguna, PCA juga punya keterbatasan yang penting dipahami.

1. PCA terutama menangkap hubungan linear

Van Der Post menekankan bahwa PCA terutama bekerja pada **linear relationships**, sehingga pola nonlinier yang lebih kompleks bisa tidak tertangkap dengan baik.

2. Interpretasi komponen kadang sulit

Karena `PC1`, `PC2`, dan seterusnya adalah kombinasi linear dari fitur asli, kadang sulit menjelaskan secara sederhana makna praktis tiap komponen. Ini juga disebut langsung di lampiran.

3. PCA sensitif terhadap skala

Tanpa scaling yang benar, PCA bisa memberi hasil yang bias ke fitur dengan rentang terbesar. Lampiran menekankan standardisasi sebagai langkah krusial sebelum PCA.

4. Tidak selalu berarti lebih baik untuk semua model

Kadang PCA membantu visualisasi dan preprocessing, tetapi bukan berarti pasti meningkatkan performa semua model. Ini harus diuji, bukan diasumsikan.

24.12 Source code Python lengkap

Berikut program lengkap yang menyatukan seluruh inti bab ini.

Simpan sebagai:

```
src/bab24_reduksi_dimensi_terpadu.py

import pandas as pd
import matplotlib.pyplot as plt
from sklearn.preprocessing import StandardScaler
from sklearn.decomposition import PCA
from sklearn.cluster import KMeans

print("=" * 80)
print("BAB 24 - UNSUPERVISED LEARNING II: REDUKSI DIMENSI")
print("=" * 80)

# 1. Load data
df = pd.read_csv("data/raw/pca_data_sederhana.csv")

print("\n[1] DATA AWAL")
print(df.head())
print("Shape:", df.shape)

# 2. Ambil fitur
X = df[["f1", "f2", "f3", "f4", "f5"]]

# 3. Scaling
scaler = StandardScaler()
X_scaled = scaler.fit_transform(X)

print("\n[2] SCALING SELESAI")
print("Contoh data hasil scaling:")
print(X_scaled[:3])

# 4. PCA ke 2 dimensi
pca = PCA(n_components=2)
X_pca = pca.fit_transform(X_scaled)

df_pca = pd.DataFrame(X_pca, columns=["PC1", "PC2"])

print("\n[3] HASIL PCA")
print(df_pca.head())

print("\nExplained variance ratio:")
print(pca.explained_variance_ratio_)
print("Total explained variance:", pca.explained_variance_ratio_.sum())

# 5. Visualisasi PCA
plt.figure(figsize=(8, 5))
plt.scatter(df_pca["PC1"], df_pca["PC2"])
plt.title("Visualisasi 2D dengan PCA")
plt.xlabel("PC1")
plt.ylabel("PC2")
plt.grid(True)
plt.tight_layout()
plt.show()
```

```
# 6. PCA + clustering
kmeans = KMeans(n_clusters=3, random_state=42, n_init=10)
cluster_labels = kmeans.fit_predict(X_scaled)
df_pca["cluster"] = cluster_labels

plt.figure(figsize=(8, 5))
plt.scatter(df_pca["PC1"], df_pca["PC2"], c=df_pca["cluster"])
plt.title("PCA 2D dengan Label Cluster")
plt.xlabel("PC1")
plt.ylabel("PC2")
plt.grid(True)
plt.tight_layout()
plt.show()

print("\n[4] INTERPRETASI AWAL")
print("- PCA mereduksi 5 fitur menjadi 2 komponen utama.")
print("- Dua komponen ini menangkap sebagian besar variasi data.")
print("- Visualisasi 2D membantu melihat pola umum dan kemungkinan
cluster.")
print("- PCA sangat berguna untuk eksplorasi, visualisasi, dan
preprocessing.")
print("=" * 80)
```

24.13 Cara membaca hasil PCA dengan sehat

Agar pembaca tidak hanya menjalankan kode, berikut cara membaca hasil PCA secara praktis.

Perhatikan explained_variance_ratio_

> Ini menunjukkan seberapa besar variasi yang dijelaskan oleh tiap komponen. Jika dua komponen pertama sudah menjelaskan proporsi besar dari total variasi, maka reduksi ke 2D cukup representatif. Lampiran Lim dan Van Der Post menekankan hal ini.

Jangan terlalu cepat menganggap PC1 dan PC2 punya arti yang jelas

> Komponen utama adalah kombinasi fitur, jadi interpretasi langsung kadang tidak sesederhana fitur asli.

Gunakan PCA terutama untuk:

- visualisasi
- eksplorasi pola
- preprocessing awal
- membantu model berbasis jarak pada dimensi tinggi

Jangan lupa konteks

> PCA sangat membantu, tetapi bukan pengganti pemahaman domain.

24.14 Kesalahan umum pemula

Ada beberapa jebakan yang sangat sering terjadi.

- **langsung PCA tanpa scaling**
  padahal lampiran secara tegas menyebut standardisasi sebagai langkah penting.
- **mengira PCA memilih fitur terbaik**
  padahal PCA membuat komponen baru, bukan sekadar memilih kolom lama.
- **menganggap semua informasi tetap utuh**
  padahal reduksi dimensi berarti ada kompromi.
- **menggunakan PCA hanya karena populer**
  padahal belum tentu selalu diperlukan.
- **menganggap PCA pasti memperbaiki semua model**
  padahal manfaatnya bergantung pada model dan data.
- **membaca visualisasi PCA terlalu berlebihan**
  padahal cluster atau pola yang terlihat tetap perlu diverifikasi lebih lanjut. Lampiran juga mengingatkan soal risiko interpretasi berlebihan dalam analisis unsupervised.

24.15 Refleksi: reduksi dimensi adalah seni merangkum tanpa terlalu banyak kehilangan

Bab ini seharusnya membuat pembaca melihat bahwa reduksi dimensi bukan trik matematis semata, tetapi cara untuk **merangkum data yang rumit menjadi bentuk yang masih bisa dipahami manusia**. PCA sangat berharga karena ia menjembatani dunia data berdimensi tinggi dengan visualisasi, eksplorasi, dan preprocessing yang lebih masuk akal. Itulah sebabnya bagian ini sangat selaras dengan lampiran yang membahas **linear regression, logistic regression, random forest, clustering, K-Means, PCA, model evaluation, dan supervised vs unsupervised learning**: PCA berdiri di tengah-tengah semuanya sebagai alat bantu yang membuat workflow data science lebih bersih dan lebih mudah dikelola.

Penutup Bab

Bab ini membahas **Unsupervised Learning II — Reduksi Dimensi** melalui pengenalan **PCA secara intuitif**, alasan **mengapa dimensi perlu dikurangi**, cara membuat **visualisasi 2D dari data berdimensi lebih tinggi**, serta **contoh source code Python** yang langsung bisa dijalankan. Seluruh pembahasan ini sangat selaras dengan materi lampiran yang menjelaskan PCA sebagai teknik reduksi dimensi berbasis komponen ortogonal yang menangkap varians terbesar, berguna untuk visualisasi, preprocessing, clustering, dan analisis data berdimensi tinggi.

Pesan terpenting dari bab ini: PCA membantu kita **melihat pola besar dalam data yang rumit**, tetapi tetap harus dipakai dengan pemahaman bahwa penyederhanaan selalu berarti ada kompromi.

Tantangan kecil untuk pembaca: modifikasi file `bab24_reduksi_dimensi_terpadu.py` agar selain `n_components=2`, Anda juga mencoba `n_components=3`, lalu bandingkan total `explained_variance_ratio_` dan lihat apakah representasi data menjadi lebih lengkap.

Bab 25. Mini Project 3 — Regresi End-to-End

Bab ini adalah titik yang sangat penting karena disinilah pembaca mulai merasakan bagaimana seluruh potongan materi sebelumnya bekerja sebagai satu alur utuh. Di buku ini, arah pembelajaran memang dibangun bertahap: dari Python dasar, data analysis, statistik praktis, workflow ML yang benar, sampai model regresi dan evaluasinya. Outline itu juga menegaskan bahwa target akhir buku adalah membuat pembaca mampu **membersihkan dan mengeksplorasi dataset nyata, membangun model klasifikasi dan regresi sederhana–menengah, membaca metrik evaluasi dengan benar, dan menghasilkan mini portofolio AI/ML**. Mini Project 3 seperti ini sangat sesuai dengan target tersebut karena memaksa pembaca menggabungkan **cleaning, feature, training, evaluasi, dan interpretasi** ke dalam satu pekerjaan yang utuh.

Materi lampiran juga sangat mendukung pendekatan ini. Referensi Van Der Post menampilkan proyek komprehensif **prediksi harga rumah** dengan beberapa model regresi, dimulai dari **loading dataset, preprocessing, scaling bila perlu, splitting, training, evaluasi, dan penarikan insight**. Referensi lain menegaskan pentingnya **split train-test**, evaluasi dengan **RMSE** dan **R^2**, serta gagasan bahwa hasil model harus dibaca sebagai bagian dari proses *feedback loop* untuk iterasi berikutnya. Dengan kata lain, kebutuhan akan mini project regresi *end-to-end* ini memang sangat relevan dengan fondasi dari lampiran.

Bab ini dirancang agar pembaca bisa menjalankan satu alur regresi lengkap:

- **dataset harga rumah / penjualan**
- **cleaning**
- **feature**
- **train model**
- **evaluasi**
- **interpretasi hasil**

Di bab ini, saya memilih **dataset harga rumah** karena sangat cocok untuk regresi, mudah dipahami secara intuitif, dan selaras dengan banyak contoh di lampiran yang juga memakai *house price prediction* sebagai studi kasus regresi.

25.1 Tujuan mini project

Tujuan utama mini project ini adalah membuat pembaca tidak berhenti pada “bisa menjalankan model”, tetapi naik ke tingkat:

- **mampu membaca masalah**
- **mampu menyiapkan data**
- **mampu memilih feature**
- **mampu membangun baseline regresi**
- **mampu mengevaluasi hasil**
- **mampu menuliskan interpretasi**

Inilah inti dari proyek AI/ML yang sehat. Materi outline buku juga berulang kali menekankan bahwa pembaca harus belajar **bukan hanya menjalankan library**, tetapi juga paham apa yang sedang dikerjakan dan dihitung.

25.2 Studi kasus: prediksi harga rumah

Skenario

Bayangkan kita bekerja sebagai analis data pemula di sebuah agen properti kecil. Kita ingin membuat model awal yang membantu memperkirakan **harga rumah** berdasarkan beberapa karakteristik dasar, misalnya:

- **luas rumah**
- **jumlah kamar**
- **usia rumah**
- **jarak ke pusat kota**
- **kondisi renovasi**

Kita tidak sedang berusaha membuat model produksi yang sempurna. Tujuan kita adalah membangun **workflow regresi end-to-end** yang masuk akal dan bisa dijelaskan.

Kenapa kasus ini cocok

Kasus ini cocok karena:

- target berupa **angka**, jadi jelas ini masalah regresi
- fitur-fitur mudah dipahami
- hubungan antarvariabel cukup intuitif
- hasil model bisa dibaca dan didiskusikan

Ini juga sangat selaras dengan contoh proyek regresi di lampiran yang memakai *house prices* sebagai studi kasus utama untuk menjelaskan regresi, regularisasi, evaluasi, dan feature importance.

25.3 Menyiapkan dataset

Agar mini project ini terasa realistis, kita akan memakai dataset yang sengaja dibuat **sedikit kotor**, supaya pembaca tetap berlatih *cleaning* sebelum modeling.

Simpan file berikut sebagai:

```
data/raw/mini_project_regresi_rumah.csv
```

Isi file:

```
id,luas_rumah,jumlah_kamar,usia_rumah,jarak_pusat_kota,kondisi_renovas
i,harga
1,80,2,20,15,tidak,250000
2,90,2,18,14,tidak,270000
3,100,3,15,12,ya,320000
4,110,3,12,11,ya,340000
5,120,3,10,10,ya,360000
6,130,4,9,9,ya,400000
7,140,4,8,8,ya,430000
8,150,4,7,7,ya,460000
9,160,5,6,6,ya,500000
10,170,5,5,5,ya,530000
11,180,5,4,4,ya,560000
12,190,6,3,3,ya,600000
13,200,6,2,2,ya,640000
14,210,6,2,2,,670000
15,220,7,1,1,ya,710000
16,230,7,1,1,ya,740000
17,240,7,1,1,ya,780000
18,250,8,1,1,ya,820000
18,250,8,1,1,ya,820000
19,260,8,0,0,ya,900000
20,dua ratus tujuh puluh,9,0,0,ya,950000
21,280,9,,0,ya,980000
22,290,9,0,,ya,1000000
23,300,10,0,0,tidak,1200000
```

Masalah yang sengaja disisipkan

Dataset ini punya beberapa masalah yang sangat cocok untuk latihan:

- ada **duplikat**
- ada nilai **kosong**
- ada tipe data salah, misalnya `luas_rumah = "dua ratus tujuh puluh"`
- ada kategori `kondisi_renovasi`
- ada kemungkinan **outlier harga**
- ada kolom `id` yang tidak relevan untuk modeling

Ini sejalan dengan bab-bab sebelumnya dan juga konsisten dengan outline yang menempatkan **cleaning, preprocessing, missing value handling, encoding, dan standardization** sebagai fondasi sebelum AI/ML.

25.4 Langkah 1 — Load dataset

Langkah pertama adalah membaca dataset dan memahami bentuk awalnya.

Simpan sebagai:

```
src/bab25_langkah1_load.py

import pandas as pd

df = pd.read_csv("data/raw/mini_project_regresi_rumah.csv")

print("=== DATA AWAL ===")
print(df.head())

print("\n=== SHAPE ===")
print(df.shape)

print("\n=== INFO ===")
print(df.info())

print("\n=== TIPE DATA ===")
print(df.dtypes)
```

Apa yang harus dilihat

Pada tahap ini, pembaca sebaiknya memeriksa:

- jumlah baris dan kolom
- kolom mana yang masih bertipe `object`
- apakah ada missing value
- apakah semua kolom numerik sudah benar-benar numerik

Kebiasaan ini juga sangat konsisten dengan pendekatan proyek regresi di lampiran, yang memulai dari **load dataset**, eksplorasi awal, dan pemeriksaan kualitas data.

25.5 Langkah 2 — Cleaning

Tahap cleaning adalah fondasi mini project ini. Kita akan melakukan:

- hapus duplikat
- perbaiki tipe data
- tangani missing value
- cek outlier sederhana
- siapkan dataset bersih untuk modeling

Source code Python lengkap

Simpan sebagai:

src/bab25_langkah2_cleaning.py

```
import pandas as pd

# 1. Load data
df =
pd.read_csv("data/raw/mini_project_regresi_rumah.csv")

print("=== SHAPE AWAL ===")
print(df.shape)

# 2. Hapus duplikat
df = df.drop_duplicates()

print("\n=== SETELAH HAPUS DUPLIKAT ===")
print(df.shape)

# 3. Perbaiki tipe data
df["luas_rumah"] = pd.to_numeric(df["luas_rumah"],
errors="coerce")
df["jumlah_kamar"] = pd.to_numeric(df["jumlah_kamar"],
errors="coerce")
df["usia_rumah"] = pd.to_numeric(df["usia_rumah"],
errors="coerce")
df["jarak_pusat_kota"] =
pd.to_numeric(df["jarak_pusat_kota"], errors="coerce")
df["harga"] = pd.to_numeric(df["harga"],
errors="coerce")

print("\n=== MISSING VALUE ===")
print(df.isnull().sum())

# 4. Isi missing value numerik dengan median
for kolom in ["luas_rumah", "jumlah_kamar",
"usia_rumah", "jarak_pusat_kota", "harga"]:
    df[kolom] = df[kolom].fillna(df[kolom].median())

# 5. Isi missing kategorikal dengan mode
df["kondisi_renovasi"] =
df["kondisi_renovasi"].fillna(df["kondisi_renovasi"].mod
e()[0])

# 6. Cek outlier sederhana pada harga dengan IQR
Q1 = df["harga"].quantile(0.25)
Q3 = df["harga"].quantile(0.75)
```

```
IQR = Q3 - Q1

batas_bawah = Q1 - 1.5 * IQR
batas_atas = Q3 + 1.5 * IQR

outlier = df[(df["harga"] < batas_bawah) | (df["harga"]
> batas_atas)]

print("\n=== OUTLIER HARGA ===")
print(outlier)

# 7. Simpan versi bersih
df.to_csv("outputs/mini_project_regresi_rumah_bersih.csv
", index=False)

print("\nFile bersih disimpan ke
outputs/mini_project_regresi_rumah_bersih.csv")
print(df.head())
```

Catatan penting

Pada tahap ini, pembaca belajar bahwa proyek regresi yang sehat selalu dimulai dari pertanyaan:

- apakah data sudah layak dimodelkan?
- adakah nilai aneh yang perlu diperiksa?
- adakah tipe data yang salah?
- apakah missing value sudah ditangani?

Ini selaras dengan penekanan di outline maupun lampiran tentang **preprocessing** dan **cleaning** sebelum modeling.

25.6 Langkah 3 — Feature engineering dasar

Sekarang kita siapkan fitur yang akan dipakai model.

Prinsip memilih feature

Untuk kasus ini, fitur yang relevan adalah:

- `luas_rumah`
- `jumlah_kamar`
- `usia_rumah`
- `jarak_pusat_kota`
- `kondisi_renovasi`

Sedangkan fitur yang **tidak dipakai**:

- `id`

Karena `id` hanyalah identitas, bukan faktor yang menjelaskan harga rumah.

Kita juga perlu:

- melakukan encoding untuk kondisi_renovasi
- membuat satu fitur turunan sederhana agar proyek terasa lebih hidup

Contoh fitur turunan

Misalnya:

- `luas_per_kamar = luas_rumah / jumlah_kamar`

Ini bisa memberi sinyal tambahan tentang kepadatan ruang.

Source code Python lengkap

Simpan sebagai:

```
src/bab25_langkah3_feature.py

import pandas as pd

df = pd.read_csv("outputs/mini_project_regresi_rumah_bersih.csv")

# 1. Feature turunan
df["luas_per_kamar"] = df["luas_rumah"] / df["jumlah_kamar"]

# 2. Encoding kategori
df["kondisi_renovasi_encoded"] = df["kondisi_renovasi"].map({
    "tidak": 0,
    "ya": 1
})

# 3. Pilih fitur akhir
fitur = [
    "luas_rumah",
    "jumlah_kamar",
    "usia_rumah",
    "jarak_pusat_kota",
    "kondisi_renovasi_encoded",
    "luas_per_kamar"
]
X = df[fitur]
y = df["harga"]

print("=== FITUR YANG DIPAKAI ===")
print(X.head())

print("\n=== TARGET ===")
print(y.head())
```

Kenapa langkah ini penting

Lampiran dan outline sama-sama menekankan bahwa *feature engineering* adalah salah satu kunci kualitas model. Model hanya bisa belajar dari fitur yang kita sediakan, sehingga pemilihan fitur yang logis sering lebih penting daripada langsung lompat ke algoritma rumit.

25.7 Langkah 4 — Train model

Untuk mini project ini, kita akan mulai dari **Linear Regression** sebagai baseline yang sehat. Ini sangat sesuai dengan lampiran yang menjadikan *Linear Regression* sebagai titik masuk alami untuk proyek prediksi harga rumah.

Source code Python lengkap

Simpan sebagai:

```
src/bab25_langkah4_train_model.py

import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LinearRegression

df = pd.read_csv("outputs/mini_project_regresi_rumah_bersih.csv")

# Feature engineering
df["luas_per_kamar"] = df["luas_rumah"] / df["jumlah_kamar"]
df["kondisi_renovasi_encoded"] = df["kondisi_renovasi"].map({
    "tidak": 0,
    "ya": 1
})

fitur = [
    "luas_rumah",
    "jumlah_kamar",
    "usia_rumah",
    "jarak_pusat_kota",
    "kondisi_renovasi_encoded",
    "luas_per_kamar"
]
X = df[fitur]
y = df["harga"]

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

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

print("=== PARAMETER MODEL ===")
```

```
print("Intercept:", model.intercept_)
print("Koefisien:")
for fitur, coef in zip(X.columns, model.coef_):
    print(f"{fitur}: {coef}")
```

Apa yang dipelajari di tahap ini

Pembaca belajar bahwa model regresi bukan “kotak hitam” penuh misteri. Setidaknya untuk Linear Regression, kita masih bisa membaca:

- **intercept**
- **koefisien**
- arah pengaruh tiap fitur

Ini salah satu kekuatan utama model linear sebagai baseline.

25.8 Langkah 5 — Evaluasi

Setelah model dilatih, kita harus menilainya. Lampiran menekankan bahwa evaluasi regresi yang baik biasanya memakai kombinasi:

- **MAE**
- **MSE**
- **RMSE**
- **R^2**

serta residual analysis sebagai bagian dari *feedback loop* untuk perbaikan model.

Source code Pytho n lengkap

Simpan sebagai:

src/bab25_langkah5_evaluasi.py

```
import pandas as pd
import math
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LinearRegression
from sklearn.metrics import mean_absolute_error,
mean_squared_error, r2_score

df =
pd.read_csv("outputs/mini_project_regresi_rumah_bersih.csv")

# Feature engineering
df["luas_per_kamar"] = df["luas_rumah"] / df["jumlah_kamar"]
df["kondisi_renovasi_encoded"] = df["kondisi_renovasi"].map({
    "tidak": 0,
    "ya": 1
})

fitur = [
    "luas_rumah",
    "jumlah_kamar",
    "usia_rumah",
    "jarak_pusat_kota",
    "kondisi_renovasi_encoded",
    "luas_per_kamar"
]

X = df[fitur]
y = df["harga"]

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

model = LinearRegression()
model.fit(X_train, y_train)

y_pred = model.predict(X_test)

mae = mean_absolute_error(y_test, y_pred)
mse = mean_squared_error(y_test, y_pred)
rmse = math.sqrt(mse)
```

```
r2 = r2_score(y_test, y_pred)

print("=== EVALUASI MODEL ===")
print("MAE  :", mae)
print("MSE  :", mse)
print("RMSE :", rmse)
print("R²   :", r2)

hasil = pd.DataFrame({
    "aktual": y_test.values,
    "prediksi": y_pred
})

hasil["residual"] = hasil["aktual"] - hasil["prediksi"]
hasil["abs_error"] = hasil["residual"].abs()

print("\n=== HASIL PREDIKSI ===")
print(hasil)

print("\n=== ERROR TERBESAR ===")
print(hasil.sort_values(by="abs_error",
ascending=False).head())
```

Apa yang harus dibaca

- **MAE** → rata-rata meleset berapa
- **RMSE** → error tipikal, lebih sensitif pada kesalahan besar
- **R²** → seberapa besar variasi harga rumah dijelaskan model
- **Residual** → di mana model paling banyak salah

Ini sangat sesuai dengan materi evaluasi regresi di lampiran.

25.9 Langkah 6 — Interpretasi hasil

Ini bagian yang sangat penting. Banyak pemula berhenti di angka metrik, padahal proyek regresi yang baik harus bisa menjawab:

- fitur mana yang paling berpengaruh
- arah pengaruhnya bagaimana
- apakah model cukup masuk akal
- apa insight awal yang bisa ditulis

Cara membaca koefisien secara hati-hati

Misalnya jika koefisien `luas_rumah` positif, maka:

semakin besar luas rumah, harga cenderung naik

Jika koefisien `jarak_pusat_kota` negatif, maka:

semakin jauh dari pusat kota, harga cenderung turun

Jika `kondisi_renovasi_encoded` positif, maka:

rumah yang sudah direnovasi cenderung bernilai lebih tinggi dibanding rumah yang belum direnovasi

Catatan penting

Interpretasi ini harus dilakukan dengan hati-hati:

- Linear Regression memberi hubungan berbasis data latih
- ini bukan bukti sebab-akibat mutlak
- hasil tetap harus dibaca dalam konteks dataset

Contoh narasi interpretasi

> Model regresi linear menunjukkan bahwa **luas rumah** dan **jumlah kamar** berasosiasi positif dengan harga rumah, yang berarti rumah yang lebih besar cenderung memiliki harga lebih tinggi. Sebaliknya, **jarak ke pusat kota** cenderung berasosiasi negatif, sehingga rumah yang lebih jauh dari pusat kota memiliki kecenderungan harga lebih rendah. Variabel **kondisi renovasi** juga memberi sinyal bahwa rumah yang sudah direnovasi cenderung bernilai lebih tinggi. Dari sisi evaluasi, model memberikan performa awal yang cukup baik sebagai baseline, namun masih terdapat residual pada beberapa observasi, yang menunjukkan bahwa belum semua variasi harga rumah dijelaskan sempurna oleh fitur yang dipakai.

Narasi seperti ini jauh lebih berharga daripada sekadar menuliskan angka metrik.

25.10 Source code Python lengkap: mini project

Berikut versi lengkap mini project ini dalam satu file.

Simpan sebagai:

```
src/bab25_mini_project_regresi_end_to_end.py

import pandas as pd
import math
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LinearRegression
from sklearn.metrics import mean_absolute_error,
mean_squared_error, r2_score

print("=" * 80)
print("BAB 25 - MINI PROJECT 3: REGRESI END-TO-END")
print("=" * 80)

# ============================================================
# 1. LOAD DATASET
# ============================================================
df = pd.read_csv("data/raw/mini_project_regresi_rumah.csv")

print("\n[1] DATA AWAL")
print(df.head())
print("Shape awal:", df.shape)

# ============================================================
# 2. CLEANING
# ============================================================
print("\n[2] CLEANING")

df = df.drop_duplicates()

for kolom in ["luas_rumah", "jumlah_kamar", "usia_rumah",
"jarak_pusat_kota", "harga"]:
    df[kolom] = pd.to_numeric(df[kolom], errors="coerce")

for kolom in ["luas_rumah", "jumlah_kamar", "usia_rumah",
"jarak_pusat_kota", "harga"]:
    df[kolom] = df[kolom].fillna(df[kolom].median())

df["kondisi_renovasi"] =
df["kondisi_renovasi"].fillna(df["kondisi_renovasi"].mode()[0
])

print("Shape setelah cleaning:", df.shape)
```

```
print("\nMissing value setelah cleaning:")
print(df.isnull().sum())

# ============================================================
# 3. FEATURE ENGINEERING
# ============================================================
print("\n[3] FEATURE ENGINEERING")

df["luas_per_kamar"] = df["luas_rumah"] / df["jumlah_kamar"]
df["kondisi_renovasi_encoded"] = df["kondisi_renovasi"].map({
    "tidak": 0,
    "ya": 1
})

fitur = [
    "luas_rumah",
    "jumlah_kamar",
    "usia_rumah",
    "jarak_pusat_kota",
    "kondisi_renovasi_encoded",
    "luas_per_kamar"
]

X = df[fitur]
y = df["harga"]

print("Fitur akhir:")
print(X.head())

# ============================================================
# 4. TRAIN MODEL
# ============================================================
print("\n[4] TRAIN MODEL")

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

model = LinearRegression()
model.fit(X_train, y_train)

print("Intercept:", model.intercept_)
print("Koefisien:")
for fitur, coef in zip(X.columns, model.coef_):
    print(f"{fitur}: {coef}")
```

```
# ============================================================
# 5. EVALUASI
# ============================================================
print("\n[5] EVALUASI")

y_pred = model.predict(X_test)

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

print("MAE  :", mae)
print("MSE  :", mse)
print("RMSE :", rmse)
print("R²   :", r2)

hasil = pd.DataFrame({
    "aktual": y_test.values,
    "prediksi": y_pred
})

hasil["residual"] = hasil["aktual"] - hasil["prediksi"]
hasil["abs_error"] = hasil["residual"].abs()

print("\nHasil prediksi:")
print(hasil)

print("\nError terbesar:")
print(hasil.sort_values(by="abs_error",
ascending=False).head())

# ============================================================
# 6. INTERPRETASI HASIL
# ============================================================
print("\n[6] INTERPRETASI HASIL")
print("- Luas rumah biasanya berasosiasi positif dengan
harga.")
print("- Jumlah kamar yang lebih banyak cenderung mendukung
harga lebih tinggi.")
print("- Jarak ke pusat kota yang makin jauh cenderung
menekan harga.")
print("- Kondisi renovasi memberi sinyal nilai tambah pada
rumah.")
print("- Model ini cocok sebagai baseline awal untuk prediksi
harga rumah.")

# ============================================================
```

```
# 7. SIMPAN DATA BERSIH
# ============================================================
df.to_csv("outputs/bab25_regresi_rumah_bersih.csv",
index=False)
print("\nData bersih disimpan di
outputs/bab25_regresi_rumah_bersih.csv")

print("\nSELESAI. Anda sudah menjalankan mini project regresi
end-to-end.")
print("=" * 80)
```

25.11 Apa yang sebaiknya ditulis di laporan atau notebook

Mini project seperti ini sebaiknya tidak berhenti di output terminal. Pembaca sebaiknya menulis narasi singkat yang memuat:

- tujuan analisis
- kondisi awal dataset
- masalah kualitas data yang ditemukan
- langkah cleaning
- fitur yang dipilih
- model yang dipakai
- metrik evaluasi
- interpretasi koefisien
- insight awal
- batasan model

Contoh narasi singkat

> Mini project ini bertujuan membangun model regresi awal untuk memprediksi harga rumah berdasarkan karakteristik dasar properti. Dataset awal mengandung beberapa masalah seperti duplikasi, missing value, dan tipe data yang tidak konsisten, sehingga dilakukan proses cleaning terlebih dahulu. Setelah data dibersihkan, dibuat beberapa fitur yang relevan, termasuk variabel turunan `luas_per_kamar` dan encoding pada kondisi renovasi. Model yang dipakai adalah Linear Regression sebagai baseline yang mudah dijelaskan. Hasil evaluasi menunjukkan bahwa model mampu menjelaskan sebagian besar variasi harga rumah dengan error yang cukup masuk akal untuk baseline awal. Koefisien model juga memberi insight bahwa luas rumah dan kondisi renovasi cenderung meningkatkan harga, sementara jarak ke pusat kota cenderung menurunkannya. Namun demikian, model ini masih sederhana dan dapat dikembangkan lebih lanjut dengan fitur tambahan atau model yang lebih kompleks.

Narasi seperti ini sangat cocok untuk portofolio awal AI/ML.

25.12 Kesalahan umum pemula

Ada beberapa jebakan yang sangat sering muncul dalam mini project regresi.

- **langsung modeling tanpa cleaning**
- **memasukkan kolom identitas seperti `id` ke model**
- **tidak melakukan encoding fitur kategorikal**
- **hanya membaca R^2 tanpa melihat error**
- **tidak memeriksa residual**
- **menganggap hasil model sebagai sebab-akibat mutlak**
- **tidak menulis interpretasi hasil**

Kebiasaan yang lebih baik adalah selalu bertanya:

> **apakah workflow saya sudah cukup sehat, dan apakah hasil model saya bisa dijelaskan?**

25.13 Refleksi: mini project regresi adalah jembatan dari belajar ke bekerja

Bab ini seharusnya membuat pembaca merasa bahwa machine learning bukan lagi kumpulan potongan materi yang terpisah. Dalam mini project seperti ini, semuanya bertemu:

- Python
- Pandas
- cleaning
- feature engineering
- train-test split
- regresi
- evaluasi
- interpretasi

Itulah kenapa mini project regresi *end-to-end* sangat penting. Lampiran juga sangat konsisten menunjukkan bahwa proyek semacam ini adalah cara terbaik untuk melatih kebiasaan kerja data science yang sehat: **load data → preprocessing → split → train → evaluate → interpret**.

Penutup Bab

Bab ini membahas **Mini Project 3 — Regresi End-to-End** melalui studi kasus **dataset harga rumah**, dengan alur **cleaning, feature, train model, evaluasi, dan interpretasi hasil**. Struktur seperti ini sangat relevan dengan outline buku utama yang menargetkan pembaca mampu bekerja dengan data nyata dan membangun model regresi sederhana–menengah, serta sangat selaras dengan materi lampiran tentang proyek regresi, preprocessing, regularisasi, evaluasi model, dan pentingnya insight yang bisa dijelaskan.

Pesan terpenting dari bab ini: mini project regresi yang baik bukan sekadar membuat model yang bisa memprediksi, tetapi membuat **workflow yang rapi, hasil yang bisa dievaluasi, dan insight yang bisa dijelaskan**.

Tantangan kecil untuk pembaca: modifikasi file `bab25_mini_project_regresi_end_to_end.py` agar selain `LinearRegression`, Anda juga mencoba `Ridge` dan `Lasso`, lalu bandingkan perubahan **RMSE**, **R^2**, dan koefisien fitur ketiganya.

Bab 26. Mini Project 4 — Klasifikasi End-to-End

Bab ini adalah pasangan alami dari mini project regresi. Jika pada proyek sebelumnya pembaca membangun **regresi end-to-end**, maka di bab ini pembaca akan menjalankan **klasifikasi end-to-end**: dari data mentah sampai memilih model terbaik. Ini sangat selaras dengan outline buku utama yang menargetkan pembaca mampu **membangun model klasifikasi dan regresi sederhana–menengah**, **membaca metrik evaluasi dengan benar**, dan menghasilkan **mini portofolio AI/ML** yang nyata.

Kebutuhan akan mini project klasifikasi seperti ini juga sangat didukung oleh materi lampiran. Salah satu referensi menampilkan proyek klasifikasi lengkap yang meliputi **loading dataset, preprocessing, feature engineering, train-test split, training model, evaluasi dengan accuracy, classification report, dan confusion matrix**. Referensi lain juga menekankan bahwa **Logistic Regression** dapat dipakai sebagai baseline, lalu dibandingkan dengan model lain seperti **Decision Tree** dan **Random Forest**, sementara evaluasi yang sehat harus melampaui accuracy saja.

Bab ini dirancang agar pembaca menjalankan alur klasifikasi yang utuh:

- **dataset pendidikan / churn / kelulusan**
- **preprocessing**
- **training beberapa model**
- **bandingkan hasil**
- **pilih model terbaik**

Agar tetap mudah dan relevan untuk pemula, saya memilih **dataset kelulusan mahasiswa**. Kasus ini sederhana, intuitif, dan sangat cocok untuk memperkenalkan workflow klasifikasi secara bersih.

26.1 Tujuan mini project

Mini project ini bertujuan membuat pembaca mampu:

- memuat dataset klasifikasi nyata sederhana
- membersihkan dan menyiapkan data
- melakukan *encoding* dan *scaling* bila perlu
- melatih beberapa model klasifikasi
- membandingkan hasil model dengan metrik yang benar
- memilih model terbaik secara argumentatif, bukan sekadar “feeling”

Ini sejalan dengan pola yang sangat ditekankan di lampiran: workflow ML yang sehat harus bergerak dari **preprocessing → split → training → evaluation → comparison**.

26.2 Studi kasus: prediksi kelulusan mahasiswa

Skenario

Misalkan sebuah program studi ingin membuat model awal untuk memprediksi apakah seorang mahasiswa **akan lulus atau tidak** berdasarkan beberapa indikator akademik sederhana.

Fitur yang tersedia

- `jam_belajar`
- `kehadiran`
- `nilai_tugas`
- `nilai_uts`
- `ikut_bimbingan`
- `akses_internet`

Target

- `lulus`
  - `0` = tidak lulus
  - `1` = lulus

Kenapa kasus ini cocok

Kasus ini cocok karena:

- targetnya jelas berupa **kelas**
- fitur mudah dipahami
- cocok untuk beberapa model dasar
- evaluasi bisa dibaca secara intuitif dengan **accuracy, precision, recall, F1-score**, dan **confusion matrix**

Materi lampiran juga sangat mendukung alur seperti ini. Salah satu contoh proyek klasifikasi lengkap memakai preprocessing, feature engineering, splitting, lalu membandingkan model-model klasifikasi.

26.3 Menyiapkan dataset

Agar mini project ini tetap realistis, dataset dibuat **sedikit kotor**, supaya pembaca tetap berlatih preprocessing sebelum modeling.

Simpan file berikut sebagai:

```
data/raw/mini_project_klasifikasi_kelulusan.csv
```

Isi file:

```
id,jam_belajar,kehadiran,nilai_tugas,nilai_uts,ikut_bimbingan,akses_in
ternet,lulus
1,2,70,60,58,tidak,ya,0
2,3,72,62,60,tidak,ya,0
3,4,75,65,64,tidak,ya,0
4,5,78,68,66,tidak,ya,0
5,6,80,72,71,ya,ya,1
6,7,84,76,75,ya,ya,1
7,8,87,80,79,ya,ya,1
8,9,90,84,83,ya,ya,1
9,10,92,87,86,ya,ya,1
10,11,95,90,89,ya,ya,1
11,4,74,64,63,tidak,tidak,0
12,5,77,67,65,tidak,ya,0
13,6,81,73,72,ya,ya,1
14,7,85,77,76,ya,,1
15,8,88,81,80,ya,ya,1
16,9,91,85,84,ya,ya,1
17,10,93,88,87,ya,ya,1
18,11,96,91,90,ya,ya,1
18,11,96,91,90,ya,ya,1
19,12,98,94,93,ya,ya,1
20,dua belas,97,95,94,ya,ya,1
21,3,,61,59,tidak,tidak,0
22,6,82,,73,ya,ya,1
23,7,86,78,,ya,ya,1
24,2,68,58,55,tidak,tidak,0
```

Masalah yang sengaja disisipkan

Dataset ini punya beberapa masalah untuk latihan preprocessing:

- ada **duplikat**
- ada nilai **kosong**
- ada tipe data salah, misalnya `jam_belajar = "dua belas"`
- ada fitur kategori seperti `ikut_bimbingan` dan `akses_internet`
- ada kolom `id` yang tidak relevan untuk model

Ini sangat sejalan dengan penekanan lampiran tentang pentingnya **data cleaning**, **encoding**, dan **train-test split** sebelum melatih model.

26.4 Langkah 1 — Load dataset

Langkah pertama adalah membaca dataset dan memahami kondisinya.

Simpan sebagai:

```
src/bab26_langkah1_load.py
```

```
import pandas as pd

df = pd.read_csv("data/raw/mini_project_klasifikasi_kelulusan.csv")

print("=== DATA AWAL ===")
print(df.head())

print("\n=== SHAPE ===")
print(df.shape)

print("\n=== INFO ===")
print(df.info())

print("\n=== TIPE DATA ===")
print(df.dtypes)
```

Apa yang harus dilihat

Pada tahap ini, pembaca sebaiknya memeriksa:

- jumlah baris dan kolom
- kolom bertipe `object`
- missing value
- potensi duplikat
- kolom yang mungkin tidak relevan untuk modeling

Kebiasaan ini konsisten dengan workflow klasifikasi pada lampiran.

26.5 Langkah 2 — Preprocessing

Tahap preprocessing di proyek klasifikasi ini mencakup:

- hapus duplikat
- perbaiki tipe data
- tangani missing value
- encoding fitur kategori
- siapkan fitur akhir

Source code Python lengkap

Simpan sebagai:

src/bab26_langkah2_preprocessing.py

```
import pandas as pd

# 1. Load data
df = pd.read_csv("data/raw/mini_project_klasifikasi_kelulusan.csv")

print("=== SHAPE AWAL ===")
print(df.shape)

# 2. Hapus duplikat
df = df.drop_duplicates()

print("\n=== SETELAH HAPUS DUPLIKAT ===")
print(df.shape)

# 3. Perbaiki tipe data numerik
for kolom in ["jam_belajar", "kehadiran", "nilai_tugas", "nilai_uts",
"lulus"]:
    df[kolom] = pd.to_numeric(df[kolom], errors="coerce")

print("\n=== MISSING VALUE ===")
print(df.isnull().sum())

# 4. Isi missing numerik dengan median
for kolom in ["jam_belajar", "kehadiran", "nilai_tugas", "nilai_uts"]:
    df[kolom] = df[kolom].fillna(df[kolom].median())

# 5. Isi missing kategorikal dengan mode
for kolom in ["ikut_bimbingan", "akses_internet"]:
    df[kolom] = df[kolom].fillna(df[kolom].mode()[0])

# 6. Encoding kategori
df["ikut_bimbingan_encoded"] = df["ikut_bimbingan"].map({"tidak": 0,
"ya": 1})
df["akses_internet_encoded"] = df["akses_internet"].map({"tidak": 0,
"ya": 1})

# 7. Simpan data bersih
df.to_csv("outputs/mini_project_klasifikasi_kelulusan_bersih.csv",
index=False)

print("\n=== DATA SETELAH PREPROCESSING ===")
print(df.head())

print("\nFile bersih disimpan ke
outputs/mini_project_klasifikasi_kelulusan_bersih.csv")
```

Kenapa preprocessing ini penting

Lampiran sangat jelas bahwa klasifikasi yang sehat harus dimulai dari **preprocessing**, termasuk menangani missing value, *feature engineering*, konversi kategorikal ke numerik, dan split yang benar.

26.6 Langkah 3 — Menentukan fitur dan target

Sekarang kita tetapkan fitur yang dipakai model.

Fitur yang dipakai

- `jam_belajar`
- `kehadiran`
- `nilai_tugas`
- `nilai_uts`
- `ikut_bimbingan_encoded`
- `akses_internet_encoded`

Target

- `lulus`

Catatan

Kolom `id` tidak dipakai karena hanya identitas, bukan faktor akademik.

Source code Python lengkap

Simpan sebagai:

```
src/bab26_langkah3_fitur_target.py

import pandas as pd

df =
pd.read_csv("outputs/mini_project_klasifikasi_kelulusan_bersih.csv")

fitur = [
    "jam_belajar",
    "kehadiran",
    "nilai_tugas",
    "nilai_uts",
    "ikut_bimbingan_encoded",
    "akses_internet_encoded"
]
X = df[fitur]
y = df["lulus"]

print("=== FITUR ===")
print(X.head())
```

```
print("\n=== TARGET ===")
print(y.head())
```

26.7 Langkah 4 — Training beberapa model

Di mini project ini, kita akan melatih tiga model yang sangat cocok untuk pemula:

- **Logistic Regression**
- **Decision Tree**
- **Random Forest**

Ini sangat sejalan dengan lampiran, yang menempatkan Logistic Regression sebagai baseline penting, serta Decision Tree dan Random Forest sebagai model klasifikasi tabular yang kuat dan umum dipakai.

Kenapa tiga model ini

- **Logistic Regression**
  baseline sederhana dan cukup interpretatif.
- **Decision Tree**
  mudah dijelaskan lewat aturan keputusan.
- **Random Forest**
  lebih kuat dan lebih stabil dibanding satu pohon tunggal.

Source code Python lengkap

Simpan sebagai:

```
src/bab26_langkah4_train_beberapa_model.py

import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.linear_model import LogisticRegression
from sklearn.tree import DecisionTreeClassifier
from sklearn.ensemble import RandomForestClassifier

df =
pd.read_csv("outputs/mini_project_klasifikasi_kelulusan_bersih.csv")

fitur = [
    "jam_belajar",
    "kehadiran",
    "nilai_tugas",
    "nilai_uts",
    "ikut_bimbingan_encoded",
    "akses_internet_encoded"
]

X = df[fitur]
y = df["lulus"]

X_train, X_test, y_train, y_test = train_test_split(
```

```
    X, y,
    test_size=0.25,
    random_state=42,
    stratify=y
)

# Scaling untuk Logistic Regression
scaler = StandardScaler()
X_train_scaled = scaler.fit_transform(X_train)
X_test_scaled = scaler.transform(X_test)

models = {
    "LogisticRegression": LogisticRegression(),
    "DecisionTree": DecisionTreeClassifier(max_depth=4,
random_state=42),
    "RandomForest": RandomForestClassifier(random_state=42)
}

for nama, model in models.items():
    if nama == "LogisticRegression":
        model.fit(X_train_scaled, y_train)
    else:
        model.fit(X_train, y_train)

print("Semua model berhasil dilatih.")
```

26.8 Langkah 5 — Bandingkan hasil model

Ini inti utama mini project ini. Kita tidak hanya melatih model, tetapi juga membandingkannya dengan metrik yang tepat.

Metrik yang akan dipakai:

- **Accuracy**
- **Precision**
- **Recall**
- **F1-score**

Lampiran juga menekankan bahwa evaluasi klasifikasi harus membaca precision, recall, dan F1-score, bukan hanya accuracy, apalagi ketika kelas bisa tidak seimbang.

Source code Python lengkap

Simpan sebagai:

```
src/bab26_langkah5_bandingkan_model.py

import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.linear_model import LogisticRegression
from sklearn.tree import DecisionTreeClassifier
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import accuracy_score, precision_score,
recall_score, f1_score

df =
pd.read_csv("outputs/mini_project_klasifikasi_kelulusan_bersih.csv")

fitur = [
    "jam_belajar",
    "kehadiran",
    "nilai_tugas",
    "nilai_uts",
    "ikut_bimbingan_encoded",
    "akses_internet_encoded"
]

X = df[fitur]
y = df["lulus"]

X_train, X_test, y_train, y_test = train_test_split(
    X, y,
    test_size=0.25,
    random_state=42,
    stratify=y
)

scaler = StandardScaler()
X_train_scaled = scaler.fit_transform(X_train)
X_test_scaled = scaler.transform(X_test)

models = {
    "LogisticRegression": LogisticRegression(),
    "DecisionTree": DecisionTreeClassifier(max_depth=4,
random_state=42),
    "RandomForest": RandomForestClassifier(random_state=42)
}

hasil = []

for nama, model in models.items():
    if nama == "LogisticRegression":
        model.fit(X_train_scaled, y_train)
        y_pred = model.predict(X_test_scaled)
    else:
        model.fit(X_train, y_train)
        y_pred = model.predict(X_test)

    hasil.append({
```

```
            "model": nama,
            "accuracy": accuracy_score(y_test, y_pred),
            "precision": precision_score(y_test, y_pred),
            "recall": recall_score(y_test, y_pred),
            "f1_score": f1_score(y_test, y_pred)
        })

hasil_df = pd.DataFrame(hasil).sort_values(by="f1_score",
ascending=False)

print("=== PERBANDINGAN MODEL ===")
print(hasil_df)
```

Kenapa F1-score penting

Kalau pembaca ingin memilih model yang lebih seimbang antara precision dan recall, **F1-score** sering sangat berguna. Lampiran menekankan bahwa F1-score sangat penting, terutama saat data tidak seimbang atau saat precision dan recall sama-sama penting.

26.9 Langkah 6 — Pilih model terbaik

Setelah metrik dibandingkan, kita harus memilih model terbaik. Tetapi “terbaik” tidak selalu berarti accuracy tertinggi. Lampiran juga menegaskan bahwa pemilihan model harus mempertimbangkan:

- performa metrik
- konteks masalah
- interpretabilitas
- kebutuhan praktis proyek

Cara memilih model secara sehat

Misalnya:

- jika semua metrik mirip, dan Anda butuh model mudah dijelaskan → **Logistic Regression**
- jika ingin aturan keputusan eksplisit → **Decision Tree**
- jika data tabular dan performa terbaik ada di model ensemble → **Random Forest**

Jadi, model terbaik adalah model yang **paling sesuai dengan tujuan proyek**, bukan sekadar paling besar satu angkanya.

Contoh narasi pemilihan model

Berdasarkan perbandingan metrik accuracy, precision, recall, dan F1-score, model **Random Forest** memberikan performa paling baik secara keseluruhan pada dataset ini. Namun, bila kebutuhan utama adalah interpretabilitas dan baseline yang mudah dijelaskan, **Logistic Regression** tetap sangat layak dipertahankan sebagai model pembanding. Dengan demikian, untuk tujuan

prediksi kelulusan yang lebih akurat pada dataset ini, Random Forest dipilih sebagai kandidat model terbaik, sementara Logistic Regression tetap penting sebagai baseline interpretatif.

Narasi seperti ini jauh lebih kuat daripada sekadar mengatakan “Random Forest paling bagus”.

26.10 Source code Python lengkap: mini project terpadu

Berikut versi lengkap mini project ini dalam satu file.

Simpan sebagai:

```
src/bab26_mini_project_klasifikasi_end_to_end.py

import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.linear_model import LogisticRegression
from sklearn.tree import DecisionTreeClassifier
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import (
    accuracy_score, precision_score, recall_score, f1_score,
    confusion_matrix, classification_report
)

print("=" * 80)
print("BAB 26 - MINI PROJECT 4: KLASIFIKASI END-TO-END")
print("=" * 80)

# ============================================================
# 1. LOAD DATASET
# ============================================================
df = pd.read_csv("data/raw/mini_project_klasifikasi_kelulusan.csv")

print("\n[1] DATA AWAL")
print(df.head())
print("Shape awal:", df.shape)

# ============================================================
# 2. PREPROCESSING
# ============================================================
print("\n[2] PREPROCESSING")

df = df.drop_duplicates()

for kolom in ["jam_belajar", "kehadiran", "nilai_tugas", "nilai_uts",
"lulus"]:
    df[kolom] = pd.to_numeric(df[kolom], errors="coerce")

for kolom in ["jam_belajar", "kehadiran", "nilai_tugas", "nilai_uts"]:
    df[kolom] = df[kolom].fillna(df[kolom].median())

for kolom in ["ikut_bimbingan", "akses_internet"]:
    df[kolom] = df[kolom].fillna(df[kolom].mode()[0])
```

```
df["ikut_bimbingan_encoded"] = df["ikut_bimbingan"].map({"tidak": 0, "ya":
1})
df["akses_internet_encoded"] = df["akses_internet"].map({"tidak": 0, "ya":
1})

print("Shape setelah preprocessing:", df.shape)

# ============================================================
# 3. FITUR DAN TARGET
# ============================================================
fitur = [
    "jam_belajar",
    "kehadiran",
    "nilai_tugas",
    "nilai_uts",
    "ikut_bimbingan_encoded",
    "akses_internet_encoded"
]

X = df[fitur]
y = df["lulus"]

# ============================================================
# 4. SPLIT DATA
# ============================================================
X_train, X_test, y_train, y_test = train_test_split(
    X, y,
    test_size=0.25,
    random_state=42,
    stratify=y
)

# Scaling khusus untuk Logistic Regression
scaler = StandardScaler()
X_train_scaled = scaler.fit_transform(X_train)
X_test_scaled = scaler.transform(X_test)

# ============================================================
# 5. TRAINING BEBERAPA MODEL
# ============================================================
models = {
    "LogisticRegression": LogisticRegression(),
    "DecisionTree": DecisionTreeClassifier(max_depth=4, random_state=42),
    "RandomForest": RandomForestClassifier(random_state=42)
}

hasil = []

for nama, model in models.items():
    if nama == "LogisticRegression":
        model.fit(X_train_scaled, y_train)
        y_pred = model.predict(X_test_scaled)
    else:
        model.fit(X_train, y_train)
        y_pred = model.predict(X_test)

    hasil.append({
        "model": nama,
        "accuracy": accuracy_score(y_test, y_pred),
```

```
            "precision": precision_score(y_test, y_pred),
            "recall": recall_score(y_test, y_pred),
            "f1_score": f1_score(y_test, y_pred)
        })

        print(f"\n=== {nama} ===")
        print("Confusion Matrix:")
        print(confusion_matrix(y_test, y_pred))
        print("\nClassification Report:")
        print(classification_report(y_test, y_pred))

# ============================================================
# 6. BANDINGKAN HASIL
# ============================================================
hasil_df = pd.DataFrame(hasil).sort_values(by="f1_score", ascending=False)

print("\n[3] PERBANDINGAN MODEL")
print(hasil_df)

# ============================================================
# 7. PILIH MODEL TERBAIK
# ============================================================
model_terbaik = hasil_df.iloc[0]["model"]

print("\n[4] MODEL TERBAIK")
print(f"Model terbaik berdasarkan F1-score: {model_terbaik}")

print("\n[5] INTERPRETASI")
print("- Logistic Regression cocok sebagai baseline yang mudah dijelaskan.")
print("- Decision Tree cocok jika ingin aturan keputusan yang eksplisit.")
print("- Random Forest sering lebih kuat untuk data tabular.")
print("- Pemilihan akhir harus melihat metrik dan kebutuhan interpretasi.")

# ============================================================
# 8. SIMPAN DATA BERSIH
# ============================================================
df.to_csv("outputs/bab26_klasifikasi_kelulusan_bersih.csv", index=False)
print("\nData bersih disimpan di
outputs/bab26_klasifikasi_kelulusan_bersih.csv")

print("\nSELESAI. Anda sudah menjalankan mini project klasifikasi
end-to-end.")
print("=" * 80)
```

26.11 Apa yang sebaiknya ditulis di laporan atau notebook

Mini project seperti ini sebaiknya diringkas dalam narasi yang memuat:

- tujuan analisis
- kondisi awal dataset
- masalah data yang ditemukan
- langkah preprocessing
- model yang dibandingkan
- metrik evaluasi utama
- alasan pemilihan model terbaik
- batasan model

Contoh narasi singkat

> Mini project ini bertujuan membangun model klasifikasi awal untuk memprediksi kelulusan mahasiswa berdasarkan indikator akademik sederhana. Dataset awal mengandung beberapa masalah seperti duplikasi, missing value, dan tipe data yang tidak konsisten, sehingga dilakukan preprocessing terlebih dahulu. Setelah data dibersihkan dan fitur kategorikal di-*encode*, tiga model dilatih dan dibandingkan, yaitu Logistic Regression, Decision Tree, dan Random Forest. Evaluasi dilakukan menggunakan accuracy, precision, recall, dan F1-score. Hasil perbandingan menunjukkan bahwa salah satu model memberikan performa paling baik secara keseluruhan, namun pemilihan akhir tetap mempertimbangkan keseimbangan antara performa dan interpretabilitas. Dengan demikian, mini project ini menunjukkan bahwa workflow klasifikasi yang sehat tidak berhenti pada training model, tetapi juga mencakup preprocessing yang benar, evaluasi yang tepat, dan pemilihan model yang argumentatif.

26.12 Kesalahan umum pemula

Ada beberapa jebakan yang sering muncul pada mini project klasifikasi:

- **langsung training tanpa preprocessing**
- **hanya melihat accuracy**
- **tidak membandingkan beberapa model**
- **tidak melakukan scaling untuk model yang terbantu scaling**
- **memilih model terbaik hanya dari satu angka**
- **tidak mempertimbangkan interpretabilitas**
- **tidak menuliskan alasan pemilihan model**

Lampiran juga sangat menekankan bahwa evaluasi klasifikasi harus lebih kaya daripada sekadar accuracy, dan model baseline seperti Logistic Regression tetap penting sebagai pembanding.

26.13 Refleksi: mini project klasifikasi adalah latihan berpikir seperti praktisi

Mini project ini penting karena pembaca mulai bergerak dari “belajar algoritma” menuju “mengerjakan proyek”. Di sinilah semua hal bertemu:

- data mentah
- preprocessing
- split train-test
- training model
- evaluasi
- perbandingan model
- pemilihan model terbaik

Itulah sebabnya mini project klasifikasi end-to-end sangat cocok sebagai portofolio awal. Materi lampiran juga berulang kali menunjukkan bahwa proyek klasifikasi yang baik harus mencakup preprocessing, training beberapa model, evaluasi multi-metrik, dan interpretasi hasil.

Penutup Bab

Bab ini membahas **Mini Project 4 — Klasifikasi End-to-End** melalui studi kasus **dataset kelulusan**, dengan alur **preprocessing, training beberapa model, membandingkan hasil, dan memilih model terbaik**. Struktur ini sangat relevan dengan outline buku utama yang menargetkan pembaca mampu membangun model klasifikasi yang sehat dan membaca metrik evaluasi dengan benar, serta sangat selaras dengan materi lampiran tentang proyek klasifikasi, Logistic Regression sebagai baseline, penggunaan Decision Tree dan Random Forest, dan pentingnya evaluasi yang kaya metrik.

Pesan terpenting dari bab ini: mini project klasifikasi yang baik bukan hanya soal melatih model, tetapi soal **membandingkan model secara jujur, membaca metrik dengan tepat, dan memilih model yang paling sesuai dengan tujuan proyek**.

Tantangan kecil untuk pembaca: modifikasi file `bab26_mini_project_klasifikasi_end_to_end.py` agar selain Logistic Regression, Decision Tree, dan Random Forest, Anda juga menambahkan **KNN**, lalu bandingkan apakah model yang paling tinggi accuracy-nya juga tetap paling baik dilihat dari **precision, recall, dan F1-score**.

Bab 27. Mini Project 5 — Clustering End-to-End

Bab ini sangat penting karena menjadi titik ketika pembaca tidak lagi hanya memahami **apa itu clustering**, tetapi mulai **mengerjakan proyek clustering secara utuh**. Dalam buku ini, jalur pembelajaran memang dirancang progresif: dari Python dasar, data analysis, *cleaning*, visualisasi, statistik, sampai *machine learning* inti, termasuk **unsupervised learning** seperti **K-Means**, **PCA**, dan **intuisi segmentasi pelanggan**. Mini project seperti ini sangat cocok dengan target hasil belajar buku, yaitu membuat pembaca mampu **membersihkan dan mengeksplorasi dataset nyata**, lalu membangun mini portofolio AI/ML yang bisa dijelaskan.

Materi lampiran juga sangat mendukung struktur bab ini. Beberapa referensi menjelaskan bahwa **clustering** adalah teknik *unsupervised learning* untuk mengelompokkan data berdasarkan kemiripan tanpa label, dan **K-Means** adalah salah satu algoritma paling populer untuk **segmentasi pelanggan**. Referensi lain menekankan tahapan praktis yang sangat sesuai dengan mini project ini: **data collection and preparation**, **EDA**, **scaling**, **training K-Means**, **visualisasi cluster**, lalu **interpretasi hasil untuk strategi bisnis sederhana**. Lampiran juga mengingatkan bahwa hasil clustering perlu dibaca hati-hati karena evaluasinya tidak sesederhana supervised learning, sehingga visualisasi, konteks domain, dan interpretasi menjadi sangat penting.

Bab ini dirancang agar pembaca mampu menjalankan alur lengkap berikut:

- **dataset pelanggan**
- **visualisasi**
- **K-Means**
- **interpretasi cluster**
- **insight bisnis sederhana**

Semua penjelasan ditulis agar **mudah dibaca oleh mahasiswa dan praktisi**, tetapi tetap cukup profesional untuk dijadikan dasar portofolio awal.

27.1 Tujuan mini project

Mini project ini bertujuan melatih pembaca agar mampu:

- membaca dan memahami dataset pelanggan
- menyiapkan fitur untuk clustering
- melakukan visualisasi awal
- menjalankan **K-Means**
- membaca hasil cluster
- memberi nama atau karakter pada setiap cluster
- menulis **insight bisnis sederhana** dari hasil segmentasi

Ini sangat sejalan dengan semangat lampiran yang menekankan bahwa clustering seharusnya tidak berhenti pada “nomor cluster”, tetapi diterjemahkan menjadi **segment pelanggan yang bermakna** untuk keputusan nyata.

27.2 Studi kasus: segmentasi pelanggan

Bayangkan kita membantu sebuah toko ritel sederhana yang ingin lebih memahami pelanggannya. Mereka belum punya label seperti:

- pelanggan premium
- pelanggan hemat
- pelanggan impulsif
- pelanggan pasif

Tetapi mereka punya data perilaku dasar pelanggan, misalnya:

- **pendapatan tahunan**
- **skor belanja**
- **frekuensi transaksi**

Tujuan kita bukan membuat model prediksi, melainkan **menemukan kelompok alami** di dalam data pelanggan.

Inilah inti clustering. Lampiran juga memberi contoh yang sangat mirip: bank atau institusi finansial menggunakan **K-Means** untuk mengelompokkan pelanggan berdasarkan perilaku transaksi dan skor pengeluaran, lalu memakai hasilnya untuk kampanye pemasaran yang lebih personal.

27.3 Menyiapkan dataset pelanggan

Agar mini project ini terasa realistis, kita akan memakai dataset pelanggan yang **cukup sederhana tetapi tidak terlalu "steril"**. Dataset ini masih nyaman untuk pemula, tetapi cukup kaya untuk segmentasi.

Simpan file berikut sebagai:

```
data/raw/mini_project_clustering_pelanggan.csv
```

Isi file:

```
pelanggan,pendapatan_tahunan,skor_belanja,frekuensi_transaksi
P1,15,39,4
P2,16,81,8
P3,17,6,1
P4,18,77,7
P5,35,40,5
P6,36,76,7
P7,37,6,1
P8,38,72,6
P9,60,55,6
P10,62,52,5
P11,64,59,6
P12,66,54,5
P13,90,20,2
P14,92,18,2
```

```
P15,94,22,2
P16,96,19,1
P17,58,57,6
P18,61,60,6
P19,40,41,4
P20,42,,4
P21,43,44,4
P22,44,45,5
P23,45,46,5
P23,45,46,5
P24,seratus,85,8
```

Dataset ini sengaja diberi beberapa masalah kecil agar pembaca tetap berlatih preprocessing:

- ada **missing value**
- ada **duplikat**
- ada tipe data salah, misalnya `pendapatan_tahunan = "seratus"`

Ini juga konsisten dengan outline buku yang menekankan pentingnya **cleaning, preprocessing, missing value handling, encoding, dan standardization** sebelum model AI/ML dijalankan.

27.4 Langkah 1 — Load dataset

Langkah pertama adalah membaca data dan memahami bentuk awalnya.

Simpan sebagai:

```
src/bab27_langkah1_load.py

import pandas as pd

df = pd.read_csv("data/raw/mini_project_clustering_pelanggan.csv")

print("=== DATA AWAL ===")
print(df.head())

print("\n=== SHAPE ===")
print(df.shape)

print("\n=== INFO ===")
print(df.info())

print("\n=== TIPE DATA ===")
print(df.dtypes)
```

Pada tahap ini, yang perlu diperhatikan adalah:

- jumlah baris dan kolom
- apakah ada kolom bertipe `object` padahal seharusnya numerik

- apakah ada missing value
- apakah ada duplikat

Kebiasaan ini sangat sesuai dengan langkah “data collection and preparation” dan “EDA” yang dijelaskan di lampiran untuk proyek segmentasi pelanggan.

27.5 Langkah 2 — Cleaning dan persiapan fitur

Sebelum clustering, kita perlu memastikan data cukup bersih. Ini penting karena lampiran menegaskan bahwa clustering sangat bergantung pada kualitas fitur, dan pada K-Means, *noise*, skala yang tidak seragam, serta nilai yang salah bisa sangat memengaruhi hasil. Lampiran juga menekankan bahwa K-Means **tidak secara khusus menangani outlier**, sehingga data perlu diperiksa lebih dulu.

Tahapan yang akan dilakukan

- hapus duplikat
- perbaiki tipe data numerik
- tangani missing value
- pilih fitur numerik yang relevan
- siapkan dataset untuk scaling

Simpan sebagai:

```
src/bab27_langkah2_cleaning.py

import pandas as pd

# 1. Load data
df = pd.read_csv("data/raw/mini_project_clustering_pelanggan.csv")

print("=== SHAPE AWAL ===")
print(df.shape)

# 2. Hapus duplikat
df = df.drop_duplicates()

print("\n=== SETELAH HAPUS DUPLIKAT ===")
print(df.shape)

# 3. Perbaiki tipe data
for kolom in ["pendapatan_tahunan", "skor_belanja",
"frekuensi_transaksi"]:
    df[kolom] = pd.to_numeric(df[kolom], errors="coerce")

print("\n=== MISSING VALUE ===")
print(df.isnull().sum())

# 4. Isi missing value numerik dengan median
for kolom in ["pendapatan_tahunan", "skor_belanja",
"frekuensi_transaksi"]:
```

```
    df[kolom] = df[kolom].fillna(df[kolom].median())

# 5. Simpan data bersih
df.to_csv("outputs/bab27_pelanggan_bersih.csv", index=False)

print("\n=== DATA SETELAH CLEANING ===")
print(df.head())

print("\nFile bersih disimpan ke outputs/bab27_pelanggan_bersih.csv")
```

Setelah langkah ini, pembaca sudah punya dataset yang lebih layak untuk clustering.

27.6 Langkah 3 — Visualisasi awal data pelanggan

Sebelum menjalankan K-Means, pembaca sebaiknya melihat pola awal data. Lampiran secara eksplisit menyarankan **scatter plot** untuk memahami hubungan antar fitur sebelum segmentasi dilakukan. Ini penting karena visualisasi awal sering memberi intuisi apakah ada kelompok-kelompok pelanggan yang mulai terlihat.

Untuk visualisasi awal, kita akan mulai dari dua fitur yang paling intuitif:

- `pendapatan_tahunan`
- `skor_belanja`

Simpan sebagai:

```
src/bab27_langkah3_visualisasi_awal.py

import pandas as pd
import matplotlib.pyplot as plt

df = pd.read_csv("outputs/bab27_pelanggan_bersih.csv")

plt.figure(figsize=(8, 5))
plt.scatter(df["pendapatan_tahunan"], df["skor_belanja"])
plt.title("Visualisasi Awal Pelanggan")
plt.xlabel("Pendapatan Tahunan")
plt.ylabel("Skor Belanja")
plt.grid(True)
plt.tight_layout()
plt.show()
```

Apa yang harus dilihat

Saat melihat grafik ini, pembaca bisa mulai bertanya:

- apakah ada kelompok pelanggan yang tampak terpisah
- apakah ada pelanggan pendapatan tinggi tetapi skor belanja rendah
- apakah ada kelompok pelanggan pendapatan rendah tetapi skor belanja tinggi

Pertanyaan-pertanyaan seperti ini adalah jantung dari analisis segmentasi.

27.7 Langkah 4 — Mengapa scaling penting

Lampiran menjelaskan dengan jelas bahwa pada studi kasus segmentasi pelanggan, fitur numerik sebaiknya **dinormalisasi atau distandardisasi** agar clustering lebih adil. Ini penting karena K-Means memakai konsep jarak, sehingga fitur dengan skala lebih besar bisa mendominasi hasil jika tidak distandardisasi.

Pada mini project ini, kita akan memakai tiga fitur:

- `pendapatan_tahunan`
- `skor_belanja`
- `frekuensi_transaksi`

Simpan sebagai:

```
src/bab27_langkah4_scaling.py

import pandas as pd
from sklearn.preprocessing import StandardScaler

df = pd.read_csv("outputs/bab27_pelanggan_bersih.csv")

fitur = ["pendapatan_tahunan", "skor_belanja", "frekuensi_transaksi"]
X = df[fitur]

scaler = StandardScaler()
X_scaled = scaler.fit_transform(X)

print("=== DATA SETELAH SCALING ===")
print(X_scaled[:5])
```

Langkah ini membuat kontribusi setiap fitur menjadi lebih seimbang dalam perhitungan jarak K-Means.

27.8 Langkah 5 — Menentukan jumlah cluster

Ini adalah salah satu keputusan terpenting dalam K-Means. Lampiran menekankan bahwa K-Means membutuhkan jumlah cluster `k` yang ditentukan pengguna, dan bahwa validasi serta interpretasi hasil cluster tetap penting agar hasilnya bermakna. Lampiran juga menekankan pentingnya **visual evaluation** dan **iterative experimentation** dalam clustering.

Untuk mini project ini, kita akan memakai pendekatan sederhana yang ramah pemula:

- coba beberapa nilai `k`
- lihat **inertia**
- lihat apakah hasil cluster nanti masuk akal secara visual dan bisnis

Simpan sebagai:

```
src/bab27_langkah5_elbow_method.py

import pandas as pd
import matplotlib.pyplot as plt
from sklearn.preprocessing import StandardScaler
from sklearn.cluster import KMeans

df = pd.read_csv("outputs/bab27_pelanggan_bersih.csv")

fitur = ["pendapatan_tahunan", "skor_belanja", "frekuensi_transaksi"]
X = df[fitur]

scaler = StandardScaler()
X_scaled = scaler.fit_transform(X)

inertia_values = []

for k in range(1, 8):
    model = KMeans(n_clusters=k, random_state=42, n_init=10)
    model.fit(X_scaled)
    inertia_values.append(model.inertia_)

plt.figure(figsize=(8, 5))
plt.plot(range(1, 8), inertia_values, marker="o")
plt.title("Elbow Method")
plt.xlabel("Jumlah Cluster (k)")
plt.ylabel("Inertia")
plt.grid(True)
plt.tight_layout()
plt.show()
```

Cara membaca Elbow Method

Cari titik saat penurunan inertia mulai **tidak terlalu tajam lagi**. Titik itu sering menjadi kandidat jumlah cluster yang cukup masuk akal.

Untuk mini project ini, kita akan memakai **k = 4** sebagai contoh kerja, karena sering cukup kaya untuk interpretasi segment pelanggan tanpa membuat hasil terlalu rumit.

27.9 Langkah 6 — Menjalankan K-Means

Setelah memilih `k`, kita bisa menjalankan K-Means. Lampiran menjelaskan langkah ini secara praktis: inisialisasi model, `fit`, lalu memberi label cluster ke data pelanggan. Itu sangat sesuai untuk proyek ini.

Simpan sebagai:

```
src/bab27_langkah6_kmeans.py

import pandas as pd
from sklearn.preprocessing import StandardScaler
from sklearn.cluster import KMeans

# 1. Load data bersih
df = pd.read_csv("outputs/bab27_pelanggan_bersih.csv")

# 2. Pilih fitur
fitur = ["pendapatan_tahunan", "skor_belanja", "frekuensi_transaksi"]
X = df[fitur]

# 3. Scaling
scaler = StandardScaler()
X_scaled = scaler.fit_transform(X)

# 4. K-Means
model = KMeans(n_clusters=4, random_state=42, n_init=10)
df["cluster"] = model.fit_predict(X_scaled)

print("=== HASIL CLUSTER ===")
print(df)

print("\nCentroid (skala standar):")
print(model.cluster_centers_)

df.to_csv("outputs/bab27_hasil_cluster.csv", index=False)
print("\nFile hasil cluster disimpan ke
outputs/bab27_hasil_cluster.csv")
```

Setelah langkah ini, setiap pelanggan sudah punya label cluster.

27.10 Langkah 7 — Visualisasi cluster

Visualisasi sangat penting dalam clustering, karena membantu kita memahami apakah segmentasi yang dihasilkan cukup masuk akal. Lampiran berkali-kali menekankan bahwa **visualisasi cluster** adalah alat utama untuk interpretasi, terutama pada segmentasi pelanggan.

Simpan sebagai:

```
src/bab27_langkah7_visualisasi_cluster.py

import pandas as pd
import matplotlib.pyplot as plt

df = pd.read_csv("outputs/bab27_hasil_cluster.csv")

plt.figure(figsize=(8, 5))
plt.scatter(
    df["pendapatan_tahunan"],
    df["skor_belanja"],
    c=df["cluster"]
)
plt.title("Visualisasi Cluster Pelanggan")
plt.xlabel("Pendapatan Tahunan")
plt.ylabel("Skor Belanja")
plt.grid(True)
plt.tight_layout()
plt.show()
```

Apa yang harus dibaca

Dari visualisasi ini, pembaca bisa mulai melihat:

- cluster mana yang berisi pelanggan pendapatan tinggi
- cluster mana yang berisi pelanggan belanja aktif
- apakah ada kelompok pelanggan yang “unik” dibanding lainnya

27.11 Langkah 8 — Interpretasi cluster

Ini bagian paling penting. Clustering bukan sekadar memberi angka `0`, `1`, `2`, atau `3`. Tugas analis adalah menerjemahkan cluster menjadi **karakter pelanggan**.

Salah satu cara paling sederhana adalah melihat **rata-rata fitur per cluster**.

Simpan sebagai:

```
src/bab27_langkah8_interpretasi_cluster.py

import pandas as pd

df = pd.read_csv("outputs/bab27_hasil_cluster.csv")
ringkasan = df.groupby("cluster")[[
    "pendapatan_tahunan",
    "skor_belanja",
    "frekuensi_transaksi"
]].mean()

print("=== RINGKASAN RATA-RATA PER CLUSTER ===")
print(ringkasan)
```

Cara membaca hasilnya

Misalnya nanti terlihat pola seperti:

- **Cluster A**
  pendapatan rendah, skor belanja tinggi, frekuensi tinggi
- **Cluster B**
  pendapatan tinggi, skor belanja rendah, frekuensi rendah
- **Cluster C**
  pendapatan sedang, skor belanja sedang, frekuensi sedang
- **Cluster D**
  pendapatan tinggi, skor belanja tinggi, frekuensi tinggi

Dari sini kita bisa memberi interpretasi awal seperti:

- pelanggan hemat tapi aktif
- pelanggan potensial tetapi belum aktif belanja
- pelanggan biasa
- pelanggan premium aktif

Referensi buku juga menekankan bahwa hasil clustering menjadi bernilai saat diterjemahkan menjadi **profil pelanggan** yang bisa dipakai untuk strategi pemasaran, pengembangan produk, atau retensi pelanggan.

27.12 Langkah 9 — Insight bisnis sederhana

Setelah cluster dipahami, kita bisa menulis insight bisnis awal. Ini sangat penting karena tujuan akhir proyek clustering bukan hanya struktur matematis, tetapi **aksi yang lebih cerdas**. Lampiran mencontohkan bahwa segmentasi pelanggan dapat dipakai untuk:

- **personalized marketing**
- **product development**
- **customer retention**
- layanan yang lebih sesuai kebutuhan pelanggan

Contoh insight bisnis sederhana

Berikut contoh insight yang bisa ditulis:

- **Cluster dengan pendapatan tinggi tetapi skor belanja rendah**
  bisa menjadi target kampanye aktivasi atau promosi premium.
- **Cluster dengan pendapatan menengah dan frekuensi tinggi**
  bisa cocok untuk program loyalitas.
- **Cluster dengan pendapatan rendah tetapi skor belanja tinggi**
  menarik untuk promosi produk bernilai kecil-menengah yang sering dibeli.
- **Cluster dengan pendapatan tinggi dan skor belanja tinggi**
  bisa dianggap sebagai pelanggan inti atau pelanggan premium.

Contoh narasi singkat

> Hasil clustering menunjukkan bahwa pelanggan tidak homogen. Terdapat beberapa segmen dengan pola pendapatan, skor belanja, dan frekuensi transaksi yang berbeda. Salah satu segmen memperlihatkan pendapatan tinggi tetapi aktivitas belanja relatif rendah, sehingga layak menjadi target kampanye aktivasi. Segmen lain menunjukkan kombinasi pendapatan dan skor belanja yang tinggi, yang dapat diposisikan sebagai pelanggan premium untuk strategi retensi dan penawaran eksklusif. Dengan demikian, hasil clustering ini dapat menjadi dasar awal untuk pemasaran yang lebih terarah dan efisien.

Narasi seperti ini sangat cocok untuk laporan mini project.

27.13 Source code Python lengkap: mini project terpadu

Berikut versi lengkap mini project clustering *end-to-end* dalam satu file.

Simpan sebagai:

```
src/bab27_mini_project_clustering_end_to_end.py

import pandas as pd
import matplotlib.pyplot as plt
from sklearn.preprocessing import StandardScaler
from sklearn.cluster import KMeans

print("=" * 80)
print("BAB 27 - MINI PROJECT 5: CLUSTERING END-TO-END")
print("=" * 80)

# ============================================================
# 1. LOAD DATASET
# ============================================================
df = pd.read_csv("data/raw/mini_project_clustering_pelanggan.csv")

print("\n[1] DATA AWAL")
print(df.head())
print("Shape awal:", df.shape)

# ============================================================
# 2. CLEANING
# ============================================================
print("\n[2] CLEANING")

df = df.drop_duplicates()

for kolom in ["pendapatan_tahunan", "skor_belanja", "frekuensi_transaksi"]:
    df[kolom] = pd.to_numeric(df[kolom], errors="coerce")

for kolom in ["pendapatan_tahunan", "skor_belanja", "frekuensi_transaksi"]:
    df[kolom] = df[kolom].fillna(df[kolom].median())

print("Shape setelah cleaning:", df.shape)
print("\nMissing value setelah cleaning:")
print(df.isnull().sum())
```

```
# ============================================================
# 3. VISUALISASI AWAL
# ============================================================
print("\n[3] VISUALISASI AWAL")

plt.figure(figsize=(8, 5))
plt.scatter(df["pendapatan_tahunan"], df["skor_belanja"])
plt.title("Visualisasi Awal Pelanggan")
plt.xlabel("Pendapatan Tahunan")
plt.ylabel("Skor Belanja")
plt.grid(True)
plt.tight_layout()
plt.show()

# ============================================================
# 4. FITUR DAN SCALING
# ============================================================
fitur = ["pendapatan_tahunan", "skor_belanja", "frekuensi_transaksi"]
X = df[fitur]

scaler = StandardScaler()
X_scaled = scaler.fit_transform(X)

# ============================================================
# 5. ELBOW METHOD
# ============================================================
print("\n[4] ELBOW METHOD")

inertia_values = []

for k in range(1, 8):
    model = KMeans(n_clusters=k, random_state=42, n_init=10)
    model.fit(X_scaled)
    inertia_values.append(model.inertia_)

plt.figure(figsize=(8, 5))
plt.plot(range(1, 8), inertia_values, marker="o")
plt.title("Elbow Method")
plt.xlabel("Jumlah Cluster (k)")
plt.ylabel("Inertia")
plt.grid(True)
plt.tight_layout()
plt.show()

# ============================================================
# 6. K-MEANS
# ============================================================
print("\n[5] K-MEANS")

k_final = 4
model = KMeans(n_clusters=k_final, random_state=42, n_init=10)
df["cluster"] = model.fit_predict(X_scaled)

print(df)

# ============================================================
# 7. VISUALISASI CLUSTER
# ============================================================
```

```
print("\n[6] VISUALISASI CLUSTER")

plt.figure(figsize=(8, 5))
plt.scatter(df["pendapatan_tahunan"], df["skor_belanja"], c=df["cluster"])
plt.title("Visualisasi Cluster Pelanggan")
plt.xlabel("Pendapatan Tahunan")
plt.ylabel("Skor Belanja")
plt.grid(True)
plt.tight_layout()
plt.show()

# ============================================================
# 8. INTERPRETASI CLUSTER
# ============================================================
print("\n[7] RINGKASAN CLUSTER")

ringkasan = df.groupby("cluster")[[
    "pendapatan_tahunan",
    "skor_belanja",
    "frekuensi_transaksi"
]].mean()

print(ringkasan)

# ============================================================
# 9. INSIGHT BISNIS
# ============================================================
print("\n[8] INSIGHT BISNIS SEDERHANA")
print("- Pelanggan dapat dibagi ke beberapa segmen perilaku yang berbeda.")
print("- Ada segmen pelanggan berpendapatan tinggi tetapi kurang aktif
belanja.")
print("- Ada segmen pelanggan yang aktif belanja dan layak diprioritaskan
untuk loyalitas.")
print("- Segmentasi ini bisa dipakai untuk strategi promosi yang lebih
terarah.")

# ============================================================
# 10. SIMPAN HASIL
# ============================================================
df.to_csv("outputs/bab27_hasil_cluster_final.csv", index=False)
print("\nHasil cluster disimpan ke outputs/bab27_hasil_cluster_final.csv")

print("\nSELESAI. Anda sudah menjalankan mini project clustering
end-to-end.")
print("=" * 80)
```

27.14 Apa yang sebaiknya ditulis di laporan atau notebook

Mini project ini sebaiknya diringkas dengan struktur narasi seperti ini:

- tujuan analisis
- kondisi awal dataset
- langkah cleaning
- fitur yang dipakai
- alasan scaling
- pemilihan jumlah cluster
- hasil cluster
- interpretasi tiap cluster
- insight bisnis sederhana
- keterbatasan analisis

Contoh narasi singkat

> Mini project ini bertujuan melakukan segmentasi pelanggan menggunakan teknik clustering pada data tanpa label. Dataset awal mengandung beberapa masalah seperti duplikasi, missing value, dan tipe data yang tidak konsisten, sehingga dilakukan cleaning terlebih dahulu. Tiga fitur numerik digunakan untuk clustering, yaitu pendapatan tahunan, skor belanja, dan frekuensi transaksi. Karena K-Means berbasis jarak, dilakukan standarisasi agar kontribusi setiap fitur lebih seimbang. Jumlah cluster dipilih melalui pendekatan eksploratif dengan melihat elbow method dan interpretasi hasil. Hasil clustering menunjukkan bahwa pelanggan dapat dibagi ke dalam beberapa segmen dengan perilaku belanja yang berbeda. Segmentasi ini memberi dasar awal untuk strategi pemasaran yang lebih terarah, seperti kampanye aktivasi untuk pelanggan berpendapatan tinggi tetapi kurang aktif, serta program loyalitas untuk pelanggan yang aktif berbelanja.

27.15 Kesalahan umum pemula

Ada beberapa jebakan yang sangat sering terjadi pada proyek clustering:

- **langsung menjalankan K-Means tanpa cleaning**
- **tidak melakukan scaling**
- **memilih jumlah cluster secara asal**
- **menganggap label cluster sebagai kebenaran mutlak**
- **tidak menulis interpretasi cluster**
- **tidak menerjemahkan hasil menjadi insight bisnis**

Lampiran juga menekankan bahwa clustering memerlukan **validasi yang hati-hati**, visualisasi, dan konteks domain agar hasilnya benar-benar bermakna.

27.16 Refleksi: clustering yang baik mengubah data tanpa label menjadi keputusan yang lebih cerdas

Mini project ini penting karena menunjukkan bahwa *machine learning* tidak selalu harus memprediksi label. Kadang nilai terbesar justru muncul ketika kita bisa **mengelompokkan data tanpa label**, lalu menemukan pola yang berguna untuk tindakan bisnis. Itulah inti clustering.

Lampiran juga sangat konsisten pada pesan ini: segmentasi pelanggan yang baik membantu organisasi memahami perilaku, menyusun strategi pemasaran yang lebih personal, meningkatkan retensi, dan mengembangkan produk yang lebih tepat sasaran. Dengan kata lain, clustering yang baik bukan sekadar memberi nomor segmen, tetapi memberi cara baru untuk melihat pelanggan.

Penutup Bab

Bab ini membahas **Mini Project 5 — Clustering End-to-End** melalui studi kasus **dataset pelanggan**, dengan alur **visualisasi, K-Means, interpretasi cluster, dan insight bisnis sederhana**. Seluruh alur ini sangat selaras dengan outline buku utama yang menempatkan **clustering dengan K-Means, intuisi segmentasi pelanggan, dan visualisasi cluster sederhana** sebagai bagian penting dari pembelajaran *unsupervised learning*, serta sangat didukung oleh materi lampiran tentang segmentasi pelanggan, scaling fitur, training K-Means, visualisasi cluster, dan penerjemahan hasil ke strategi bisnis.

Pesan terpenting dari bab ini: mini project clustering yang baik bukan sekadar membagi pelanggan ke beberapa kelompok, tetapi membantu kita **melihat pola perilaku yang sebelumnya tersembunyi dan menerjemahkannya menjadi insight bisnis yang bisa ditindaklanjuti**.

Tantangan kecil untuk pembaca: modifikasi file `bab27_mini_project_clustering_end_to_end.py` agar selain `k=4`, Anda juga mencoba `k=3` dan `k=5`, lalu bandingkan perubahan **visualisasi cluster**, **rata-rata tiap cluster**, dan **makna bisnis** yang muncul.

Bagian V — Penutup dan Horizon Lanjutan

Bab 28. Dari Notebook ke Portofolio

Bab ini sangat penting karena menjadi jembatan antara **belajar** dan **terlihat siap bekerja**. Di buku ini, target hasil belajar akhir tidak berhenti pada kemampuan membersihkan data, membangun model, dan membaca metrik, tetapi juga sampai pembaca mampu **membuat mini portofolio AI/ML berbasis notebook**. Pada bagian penutup outline, bahkan ditegaskan perlunya **deployment ringan dan portofolio pertama**, termasuk **struktur repo GitHub yang rapi**, **isi README yang baik**, dan **cara memilih proyek portofolio yang "menjual"**. Itu berarti bab ini bukan tambahan, melainkan salah satu hasil akhir paling penting dari seluruh perjalanan belajar di buku ini.

Materi lampiran juga sangat mendukung arah ini. Buku Jain menekankan bahwa **Jupyter Notebook** sangat cocok untuk pembelajaran dan analisis data karena mendukung **kode, Markdown, penjelasan, dan visualisasi dalam satu tempat**, serta mudah diekspor dan dibagikan. Buku Lim juga menekankan pentingnya file `*.ipynb` yang berisi komentar dan langkah-langkah jelas agar pembaca bisa mengikuti alur analisis secara *hands-on*. Sementara lampiran lain menegaskan bahwa proyek yang baik seharusnya menghasilkan **laporan proyek**, **notebook yang terdokumentasi**, dan penjelasan yang jelas tentang hasil analisis serta insight yang diperoleh.

Bab ini dirancang agar pembaca mampu:

- **menyimpan notebook yang rapi**
- **menulis README GitHub**
- **memilih project portofolio pertama**
- **membuat dokumentasi hasil analisis**

Tujuannya sederhana tetapi penting: setelah belajar Python, data, statistik, regresi, klasifikasi, clustering, dan mini project, pembaca tidak berhenti pada file yang berantakan di laptop, tetapi mampu mengubahnya menjadi **portofolio yang bisa ditunjukkan**.

28.1 Mengapa portofolio penting

Banyak pembelajar pemula merasa bahwa yang paling penting adalah "menguasai library". Padahal dalam praktik nyata, orang lain tidak bisa melihat proses belajar di kepala kita. Mereka biasanya melihat:

- proyek apa yang pernah kita kerjakan
- apakah notebook kita rapi
- apakah hasil analisis bisa dijelaskan
- apakah kita bisa menuliskan konteks, langkah, hasil, dan insight
- apakah repo GitHub kita mudah dibaca

Itulah sebabnya portofolio sangat penting. Portofolio adalah bukti bahwa Anda bukan hanya menonton tutorial, tetapi **sudah mengerjakan sesuatu sampai selesai**. Ini sangat sejalan

dengan outline utama buku yang secara eksplisit menargetkan pembaca sampai mampu membuat **mini portofolio AI/ML berbasis notebook** dan menyiapkan **portofolio pertama**.

Secara sederhana, portofolio membantu menjawab pertanyaan:

> **"Kalau saya ingin meyakinkan orang bahwa saya bisa data science/AI/ML dasar, apa buktinya?"**

Jawabannya adalah:

- notebook yang rapi
- repo GitHub yang jelas
- README yang enak dibaca
- dokumentasi hasil analisis yang bisa dipahami manusia

28.2 Dari file latihan menjadi karya yang layak ditunjukkan

Selama belajar, pembaca mungkin menghasilkan banyak file seperti:

- `test1.ipynb`
- `latihan_fix_baru_final2.ipynb`
- `script_coba.py`
- `hasil_bener_fix.py`

Itu wajar saat masih belajar. Tetapi untuk portofolio, file seperti itu harus dirapikan menjadi proyek yang punya struktur jelas dan cerita yang utuh.

Materi lampiran tentang Jupyter Notebook menegaskan bahwa notebook yang baik seharusnya menggabungkan:

- kode
- penjelasan dengan Markdown
- heading yang jelas
- visualisasi
- hasil yang mudah dibagikan

Sedangkan lampiran lain menekankan bahwa deliverable proyek yang baik biasanya mencakup:

- **project report**
- **Jupyter Notebook**
- penjelasan hasil dan insight
- dokumentasi langkah-langkah yang dilakukan

Artinya, langkah pentingnya bukan hanya "punya proyek", tetapi:

> **mengubah hasil kerja menjadi proyek yang bisa dipahami orang lain.**

28.3 Menyimpan notebook yang rapi

Ini bagian pertama yang harus dikuasai.

A. Ciri notebook yang rapi

Notebook yang rapi biasanya punya:

- judul yang jelas
- tujuan analisis
- deskripsi dataset
- langkah-langkah yang berurutan
- kode yang tidak berantakan
- visualisasi yang relevan
- interpretasi hasil
- penutup atau kesimpulan

Lampiran Jain menekankan bahwa Jupyter Notebook sangat kuat karena memungkinkan penambahan **Markdown cells** untuk penjelasan, heading, dan dokumentasi, bukan hanya kode mentah.

B. Struktur notebook yang disarankan

Untuk pemula, saya sarankan struktur seperti ini:

1. **Judul proyek**
2. **Tujuan analisis**
3. **Deskripsi dataset**
4. **Import library**
5. **Load data**
6. **Cleaning / preprocessing**
7. **EDA / visualisasi**
8. **Modeling**
9. **Evaluasi**
10. **Insight / kesimpulan**

Dengan struktur ini, notebook terasa seperti dokumen kerja profesional, bukan sekadar tempat menjalankan kode.

C. Contoh judul notebook yang baik

Daripada:

- `latihan3.ipynb`

lebih baik:

- `prediksi-harga-rumah-linear-regression.ipynb`
- `segmentasi-pelanggan-kmeans.ipynb`
- `klasifikasi-kelulusan-random-forest.ipynb`

Nama file adalah kesan pertama.

D. Hindari notebook yang terlalu “ramai”

Beberapa hal yang sebaiknya dihindari:

- sel kode kosong terlalu banyak
- output yang tidak relevan
- impor library berulang
- variabel dengan nama tidak jelas
- eksperimen yang gagal tetapi tidak dibersihkan
- banyak sel tanpa penjelasan

Notebook portofolio harus terasa seperti **proyek yang selesai**, bukan ruang latihan mentah.

E. Contoh susunan Markdown notebook

Berikut contoh isi Markdown awal yang rapi:

```
# Prediksi Harga Rumah dengan Linear Regression

## Tujuan
Notebook ini bertujuan membangun model regresi sederhana untuk
memprediksi harga rumah berdasarkan beberapa fitur utama seperti
luas rumah, jumlah kamar, dan usia rumah.

## Dataset
Dataset berisi data rumah dengan beberapa fitur numerik dan
target harga.

## Alur Analisis
1. Load dataset
2. Data cleaning
3. Feature engineering sederhana
4. Training model
5. Evaluasi model
6. Interpretasi hasil
```

Struktur seperti ini langsung membuat notebook lebih profesional.

28.4 Langkah praktis merapikan notebook

Agar pembaca bisa langsung melakukannya, berikut tahap yang saya sarankan.

Langkah 1 — pilih satu notebook yang paling layak

Jangan mulai dari semua file sekaligus. Ambil satu proyek yang paling rapi atau paling menarik.

Langkah 2 — hapus sel yang tidak perlu

Buang:

- eksperimen gagal yang tidak relevan
- kode duplikat
- output yang tidak membantu pembaca

Langkah 3 — tambahkan Markdown

Pastikan setiap bagian penting punya judul dan penjelasan singkat.

Langkah 4 — jalankan notebook dari atas ke bawah

Notebook portofolio harus bisa dijalankan berurutan dengan hasil yang konsisten.

Langkah 5 — cek hasil visualisasi

Pastikan grafik:

- punya judul
- sumbu jelas
- benar-benar membantu analisis

Langkah 6 — tambahkan kesimpulan

Jangan tutup notebook dengan sel kode terakhir. Tutup dengan insight.

Ini sangat sejalan dengan lampiran yang menekankan notebook sebagai dokumen pembelajaran dan komunikasi, bukan hanya tempat menjalankan Python.

28.5 Menulis README GitHub

Ini bagian yang sering diabaikan, padahal sangat penting. Outline utama buku secara eksplisit menyebut **isi README yang baik** sebagai bagian dari portofolio pertama.

README adalah “pintu masuk” proyek Anda. Banyak orang akan melihat README terlebih dahulu sebelum membuka notebook.

A. Fungsi README

README membantu menjawab:

- proyek ini tentang apa
- dataset-nya apa
- tools yang dipakai apa
- bagaimana menjalankannya
- hasil utamanya apa
- insight terpentingnya apa

B. Struktur README yang disarankan

Saya sarankan format sederhana seperti ini:

1. **Judul proyek**
2. **Ringkasan singkat**
3. **Tujuan proyek**
4. **Dataset**
5. **Tools / library**
6. **Langkah analisis**
7. **Hasil utama**
8. **Struktur folder**
9. **Cara menjalankan**
10. **Insight / kesimpulan**

C. Contoh template README GitHub

```
# Prediksi Harga Rumah dengan Linear Regression

## Ringkasan
Proyek ini membangun model regresi sederhana untuk memprediksi harga
rumah berdasarkan fitur seperti luas rumah, jumlah kamar, usia rumah,
dan jarak ke pusat kota.

## Tujuan
- Melakukan data cleaning pada dataset rumah
- Membangun baseline model regresi
- Mengevaluasi performa model dengan MAE, RMSE, dan R²
- Menafsirkan pengaruh fitur terhadap harga rumah

## Dataset
Dataset berisi data rumah dengan beberapa fitur numerik dan target
harga.

## Tools
- Python
- pandas
- matplotlib
- scikit-learn
- Jupyter Notebook

## Langkah Analisis
1. Load data
2. Cleaning
3. Feature engineering
4. Train-test split
5. Training model
6. Evaluasi
7. Interpretasi hasil

## Hasil Utama
Model regresi memberikan baseline prediksi harga rumah yang cukup baik
dan menunjukkan bahwa luas rumah berasosiasi positif terhadap harga,
sedangkan jarak ke pusat kota cenderung berasosiasi negatif.

## Struktur Folder
- `notebooks/` → notebook analisis
- `data/` → dataset
- `images/` → visualisasi
- `README.md` → deskripsi proyek

## Cara Menjalankan
Buka notebook di Jupyter Notebook atau VS Code, lalu jalankan sel dari
atas ke bawah.

## Kesimpulan
Proyek ini menunjukkan workflow regresi sederhana end-to-end yang
cocok untuk portofolio awal data science dan machine learning.
```

D. Tips agar README enak dibaca

- pakai kalimat pendek dan jelas
- jangan terlalu panjang
- fokus pada konteks, langkah, hasil, dan insight
- hindari jargon berlebihan
- anggap pembaca README belum tahu proyek Anda

28.6 Memilih project portofolio pertama

Ini pertanyaan yang sangat penting. Outline utama buku menyebut perlunya **cara memilih 3 proyek portofolio yang "menjual"**, dan lampiran Jain juga menyarankan beberapa jenis proyek yang cocok untuk pembelajar awal, seperti:

- **EDA**
- **predictive modeling**
- **web scraping + analysis**
- **NLP sederhana**

Untuk pembaca buku ini, proyek portofolio pertama sebaiknya memenuhi tiga syarat:

- **cukup sederhana untuk diselesaikan**
- **cukup jelas untuk dijelaskan**
- **cukup relevan untuk menunjukkan kemampuan inti**

A. Karakter proyek pertama yang baik

Pilih proyek yang:

- punya tujuan jelas
- datanya tidak terlalu rumit
- bisa selesai end-to-end
- punya notebook yang rapi
- bisa menghasilkan insight yang mudah dipahami

B. Proyek pertama yang sangat cocok

Berikut beberapa pilihan paling aman:

- **EDA pada dataset pendidikan**
  Menunjukkan kemampuan membaca data, visualisasi, statistik, dan insight.
- **Regresi harga rumah**
  Menunjukkan workflow regresi lengkap.
- **Klasifikasi kelulusan / churn sederhana**
  Menunjukkan preprocessing, training model, dan evaluasi klasifikasi.
- **Clustering segmentasi pelanggan**
  Menunjukkan *unsupervised learning* dan interpretasi bisnis.

Semua ini juga sangat sesuai dengan mini project yang sudah dibangun di outline buku.

C. Pilih proyek yang “punya cerita”

Portofolio yang bagus biasanya punya cerita seperti:

- “Saya ingin memahami faktor kelulusan mahasiswa”
- “Saya ingin memprediksi harga rumah berdasarkan fitur properti”
- “Saya ingin membagi pelanggan ke beberapa segmen”
- “Saya ingin mengeksplorasi pola penjualan ritel”

Cerita ini membuat proyek lebih hidup daripada sekadar “saya menjalankan model X pada dataset Y”.

28.7 Contoh tiga proyek portofolio awal yang “menjual”

Berdasarkan arah outline utama, saya sarankan pembaca menyiapkan minimal tiga proyek awal berikut.

1. Proyek EDA

Contoh:

- analisis dataset pendidikan
- analisis penjualan ritel
- analisis dataset Titanic

Yang ditunjukkan:

- data cleaning
- visualisasi
- statistik deskriptif
- insight awal

2. Proyek supervised learning

Contoh:

- regresi harga rumah
- klasifikasi kelulusan
- klasifikasi churn pelanggan

Yang ditunjukkan:

- preprocessing
- feature engineering
- train-test split
- training model

- evaluasi

3. Proyek unsupervised learning

Contoh:

- segmentasi pelanggan dengan K-Means
- PCA + visualisasi cluster

Yang ditunjukkan:

- clustering
- reduksi dimensi
- visualisasi
- interpretasi segmen

Ini juga sangat konsisten dengan outline utama buku yang mengarah ke kombinasi **EDA + supervised learning + unsupervised learning** sebagai fondasi portofolio pertama.

28.8 Dokumentasi hasil analisis

Ini bagian yang sering disepelekan. Padahal lampiran secara eksplisit menekankan bahwa deliverable proyek yang baik harus mencakup **penjelasan langkah, kode, hasil, dan insight**. Bahkan ada penekanan bahwa notebook harus terdokumentasi dengan baik lewat **comments** dan **markdown cells**.

A. Apa yang perlu didokumentasikan

Untuk setiap proyek, sebaiknya dokumentasikan:

- **tujuan proyek**
- **deskripsi dataset**
- **masalah kualitas data yang ditemukan**
- **langkah preprocessing**
- **fitur yang dipakai**
- **model yang diuji**
- **metrik evaluasi**
- **hasil utama**
- **insight**
- **keterbatasan analisis**

B. Dokumentasi bukan hanya untuk orang lain

Dokumentasi juga membantu Anda sendiri:

- mengingat apa yang sudah dilakukan
- mengulangi eksperimen
- memperbaiki proyek di masa depan
- menjelaskan proyek saat wawancara atau presentasi

C. Contoh format dokumentasi singkat

Misalnya untuk proyek klasifikasi:

```
## Tujuan
Memprediksi kelulusan mahasiswa berdasarkan jam belajar,
kehadiran, nilai tugas, dan nilai UTS.

## Dataset
Dataset berisi data akademik sederhana mahasiswa dengan target
lulus/tidak lulus.

## Preprocessing
- Hapus duplikat
- Isi missing value
- Encoding fitur kategorikal
- Scaling untuk Logistic Regression

## Model
- Logistic Regression
- Decision Tree
- Random Forest

## Evaluasi
- Accuracy
- Precision
- Recall
- F1-score

## Hasil
Random Forest memberi performa terbaik secara keseluruhan,
tetapi Logistic Regression tetap berguna sebagai baseline yang
lebih mudah dijelaskan.

## Insight
Jam belajar, kehadiran, dan nilai tugas memberi kontribusi kuat
terhadap kemungkinan kelulusan.
```

Format seperti ini sangat membantu membuat proyek terasa matang.

28.9 Struktur repo GitHub yang rapi

Outline utama secara eksplisit menyebut **struktur repo GitHub yang rapi**. Ini penting sekali. Repo yang baik tidak harus rumit, tetapi harus teratur.

Struktur folder sederhana yang disarankan

```
project-name/
│
├── data/
│   └── dataset.csv
├── notebooks/
│   └── analisis.ipynb
├── src/
│   └── script_pendukung.py
├── images/
│   └── visualisasi.png
├── requirements.txt
└── README.md
```

Penjelasan singkat

- `data/` → dataset
- `notebooks/` → notebook utama
- `src/` → script Python pendukung
- `images/` → hasil plot atau gambar
- `requirements.txt` → daftar library
- `README.md` → deskripsi proyek

Contoh `requirements.txt`

```
pandas
numpy
matplotlib
scikit-learn
jupyter
```

Lampiran Jain juga menekankan penggunaan `requirements.txt` melalui `pip freeze > requirements.txt` sebagai praktik yang baik untuk manajemen dependency.

28.10 Langkah praktis mengubah proyek menjadi portofolio GitHub

Agar pembaca benar-benar bisa melakukannya, berikut alur praktisnya.

Langkah 1 — pilih satu proyek terbaik

Ambil satu mini project yang paling matang.

Langkah 2 — rapikan notebook

Pastikan notebook:

- bisa dijalankan dari awal sampai akhir
- punya Markdown yang jelas
- tidak penuh sel percobaan yang kacau

Langkah 3 — siapkan folder repo

Buat struktur sederhana seperti:

- `data/`
- `notebooks/`
- `README.md`

Langkah 4 — tulis README

Buat README yang menjelaskan:

- apa proyeknya
- data apa yang dipakai
- apa hasil utamanya

Langkah 5 — tambahkan visualisasi terbaik

Pilih 1–3 plot yang paling membantu menjelaskan proyek.

Langkah 6 — unggah ke GitHub

Pastikan nama repo jelas, misalnya:

- `house-price-regression-python`
- `student-graduation-classification`
- `customer-segmentation-kmeans`

Langkah 7 — cek dari sudut pandang pembaca baru

Tanyakan:

- kalau orang lain membuka repo ini, apakah mereka paham apa yang saya kerjakan?

28.11 Contoh isi README untuk proyek portofolio pertama

Berikut contoh README yang lebih singkat dan cocok untuk pemula.

```
# Student Graduation Classification

## About This Project
This project builds a simple machine learning model to predict whether
a student will graduate or not based on study hours, attendance,
assignment scores, and midterm scores.

## Objective
The goal of this project is to practice an end-to-end classification
workflow:
- data cleaning
- preprocessing
- training multiple models
- comparing evaluation metrics
- choosing the best model

## Dataset
The dataset contains simple student academic performance features with
a binary target:
- 0 = not graduate
- 1 = graduate

## Tools
- Python
- pandas
- matplotlib
- scikit-learn
- Jupyter Notebook

## Models
- Logistic Regression
- Decision Tree
- Random Forest

## Evaluation
The models were compared using:
- accuracy
- precision
- recall
- F1-score

## Main Result
Random Forest achieved the best overall classification performance,
while Logistic Regression remained useful as a simple and
interpretable baseline.
```

```
## Project Structure
- `data/` : dataset
- `notebooks/` : analysis notebook
- `README.md` : project description

## Conclusion
This project demonstrates a complete beginner-friendly machine
learning classification workflow using Python.
```

README seperti ini sudah cukup baik untuk portofolio awa.

28.12 Peran dokumentasi visual

Portofolio menjadi jauh lebih menarik jika tidak hanya berisi kode dan teks, tetapi juga visualisasi yang relevan.

Contoh visual yang bagus untuk portofolio:

- histogram distribusi data
- scatter plot regresi
- confusion matrix
- plot prediksi vs aktual
- visualisasi cluster

Tetapi pilih visual yang **membantu menjelaskan cerita**, bukan sekadar banyak grafik. Lampiran juga menekankan pentingnya dokumentasi yang jelas dan profesional, termasuk hasil, output, dan interpretasi.

28.13 Apa yang membuat proyek portofolio terlihat “menjual”

Outline utama memakai istilah “proyek portofolio yang menjual”. Ini bisa diterjemahkan secara sederhana sebagai proyek yang:

- **jelas tujuannya**
- **rapi tampilannya**
- **mudah dipahami pembaca**
- **punya insight**
- **terlihat selesai**
- **menunjukkan workflow end-to-end**

Yang membuat proyek terlihat kuat bukan hanya modelnya, tetapi kombinasi dari:

- konteks masalah
- data
- proses
- hasil
- interpretasi

Jadi, proyek dengan **Linear Regression** yang rapi dan bisa dijelaskan sering lebih kuat daripada proyek dengan model canggih tetapi dokumentasinya berantakan.

28.14 Kesalahan umum pemula saat membangun portofolio

Ada beberapa jebakan yang sangat sering terjadi.

- **mengunggah notebook yang belum rapi**
- **tidak menulis README**
- **nama file dan repo tidak jelas**
- **terlalu banyak proyek kecil, tapi tidak ada yang selesai**
- **tidak menjelaskan insight**
- **menganggap kode saja sudah cukup**
- **mengunggah proyek tanpa konteks**

Lampiran tentang dokumentasi dan deliverable juga memperingatkan bahwa hasil proyek sebaiknya dijelaskan dengan baik, bukan hanya ditampilkan sebagai kode mentah.

28.15 Refleksi: portofolio adalah bukti bahwa Anda bisa menyelesaikan sesuatu

Bab ini seharusnya membuat pembaca menyadari satu hal penting: dunia nyata jarang menilai Anda dari berapa banyak bab yang sudah dibaca. Yang lebih sering terlihat adalah:

- apa yang sudah Anda buat
- apakah hasil kerja Anda rapi
- apakah Anda bisa menjelaskannya
- apakah Anda bisa menyelesaikan proyek dari awal sampai akhir

Itulah mengapa bab ini sangat selaras dengan target hasil belajar buku. Portofolio adalah bentuk nyata dari seluruh proses belajar yang sudah dijalani: Python, data analysis, machine learning, evaluasi, mini project, dan dokumentasi. Outline utama juga menegaskan bahwa buku ini sebaiknya ditutup dengan **portofolio dan deployment ringan**, karena hasil akhir yang terlihat akan membantu pembaca tetap termotivasi.

Penutup Bab

Bab ini membahas **Dari Notebook ke Portofolio** melalui empat hal utama: **menyimpan notebook yang rapi**, **menulis README GitHub**, **memilih project portofolio pertama**, dan **mendokumentasikan hasil analisis**. Seluruh pembahasan ini sangat selaras dengan outline utama buku yang menargetkan pembaca mampu membuat **mini portofolio AI/ML berbasis notebook** dan menyiapkan **portofolio pertama** dengan **struktur repo GitHub yang rapi** serta **README yang baik**, dan juga didukung oleh materi lampiran yang menekankan pentingnya **Jupyter Notebook**, **Markdown**, **well-documented notebooks**, serta deliverable proyek yang jelas.

Pesan terpenting dari bab ini: portofolio yang baik bukan hanya kumpulan kode, tetapi **kumpulan proyek yang rapi, bisa dijalankan, bisa dijelaskan, dan menunjukkan bahwa Anda benar-benar paham apa yang Anda kerjakan**.

Tantangan kecil untuk pembaca: pilih satu mini project terbaik dari buku ini, lalu ubah menjadi repo GitHub lengkap dengan **notebook yang rapi**, **README**, **struktur folder yang jelas**, dan **satu ringkasan insight utama** yang bisa dipahami orang lain dalam waktu kurang dari dua menit.

Bab 29. Langkah Berikutnya setelah Buku Ini

Bab ini adalah **jembatan dari fondasi ke horizon lanjutan**. Dalam buku ini, target hasil belajar akhirnya memang bukan membuat pembaca langsung menjadi ahli *deep learning* atau *LLM engineer*, tetapi membuat pembaca cukup kuat di **Python + data + ML klasik**, lalu **mengenal arah lanjutan: deep learning, LLM, AI coding assistant, dan deployment ringan**. Outline yang sama juga secara eksplisit menyarankan agar bagian lanjutan ini ditempatkan sebagai **horizon belajar berikutnya**, bukan sebagai inti teknis yang terlalu berat untuk pemula.

Arah ini juga sangat selaras dengan materi lampiran. Lampiran menjelaskan bahwa setelah fondasi ML klasik cukup kuat, pembelajar bisa mulai bergerak ke **deep learning**, **NLP dan LLM**, penggunaan **AI coding assistant** secara sehat, serta **deployment sederhana** untuk membuat hasil kerja lebih nyata. Di saat yang sama, beberapa lampiran juga mengingatkan bahwa langkah lanjutan ini harus tetap dijalani dengan sikap kritis: data tetap harus bersih, hasil model tetap harus dievaluasi, dan kode yang dihasilkan AI tetap harus diverifikasi.

Bab ini dirancang agar pembaca mendapatkan gambaran realistis tentang:

- **deep learning dasar**
- **NLP dan LLM**
- **AI coding assistant**
- **deployment sederhana**
- **roadmap belajar 6 bulan**

Tujuannya bukan membuat pembaca langsung menguasai semuanya, tetapi membantu pembaca melihat **jalan belajar yang masuk akal setelah buku ini selesai**.

29.1 Setelah fondasi ML, ke mana sebaiknya melangkah

Setelah menyelesaikan buku ini, pembaca idealnya sudah cukup nyaman dengan:

- Python dasar
- NumPy dan Pandas
- visualisasi
- statistik praktis
- workflow ML
- regresi
- klasifikasi
- clustering
- mini project
- portofolio awal

Itu sudah sangat berharga. Outline utama bahkan menekankan bahwa hasil realistis dari 4–6 bulan belajar konsisten adalah kemampuan membangun model sederhana–menengah, membaca metrik dengan benar, dan membuat mini portofolio. Jadi langkah berikutnya sebaiknya **bukan melompat liar**, tetapi **naik satu tingkat secara terarah**.

Secara praktis, ada empat pintu lanjutan yang paling masuk akal:

- **deep learning dasar**
- **NLP dan LLM**
- **AI coding assistant**
- **deployment sederhana**

Mari kita bahas satu per satu.

29.2 Deep learning dasar

A. Kenapa deep learning menjadi langkah lanjutan alami

Outline utama buku menempatkan **deep learning** sebagai salah satu arah lanjutan yang perlu dikenalkan setelah fondasi ML klasik kuat. Bahkan pada versi outline yang lain, disebut juga pengantar **neuron, layer, aktivasi, loss, perceptron, sampai MLP sederhana**, dengan penekanan bahwa untuk buku pemula, *deep learning* sebaiknya hadir sebagai **pengantar inspiratif**, bukan bahasan yang terlalu berat.

Lampiran juga menjelaskan bahwa *deep learning* sangat kuat untuk tugas-tugas kompleks seperti:

- visi komputer
- klasifikasi citra
- pemrosesan bahasa
- pengenalan pola pada data besar

Namun lampiran yang sama juga menekankan bahwa *deep learning* biasanya lebih menuntut dari sisi:

- data
- komputasi
- kompleksitas model
- interpretabilitas

B. Apa yang sebaiknya dipelajari dulu

Untuk pemula, saya sarankan urutan yang sederhana:

- pahami **bedanya ML klasik dan deep learning**
- pahami **neuron**, **layer**, dan **aktivasi**
- pelajari **MLP** untuk klasifikasi dasar
- kenali **framework modern**, terutama *PyTorch* atau *TensorFlow/Keras*

Outline utama bahkan menyinggung bahwa *PyTorch* menarik untuk pembelajaran modern.

C. Kapan mulai serius ke deep learning

Mulailah lebih serius ke *deep learning* jika:

- Anda sudah nyaman dengan evaluasi model klasik
- Anda mengerti train/validation/test
- Anda siap belajar konsep seperti *tensor*, *epochs*, *loss*, dan *backpropagation*
- Anda ingin bergerak ke image, speech, atau NLP modern

D. Sikap yang sehat

Jangan merasa tertinggal kalau belum langsung masuk *deep learning*. Fondasi ML klasik yang kuat justru membuat Anda lebih siap membaca *deep learning* dengan tenang.

29.3 NLP dan LLM

A. Kenapa NLP dan LLM penting

Outline utama buku secara eksplisit menyebut **NLP, LLM, dan prompt engineering dasar** sebagai horizon lanjutan yang sangat relevan untuk pembelajar modern. Ia juga menekankan bahwa **AI coding assistant** harus dipahami sebagai partner belajar, bukan pengganti berpikir.

Lampiran menjelaskan bahwa evolusi NLP bergerak dari sistem berbasis aturan dan fitur buatan manusia menuju pendekatan modern berbasis **machine learning** dan **deep learning**, termasuk model seperti **BERT** dan keluarga transformer lain. Lampiran lain juga menjelaskan bahwa model NLP modern dipakai untuk:

- klasifikasi teks
- analisis sentimen
- rangkuman
- terjemahan
- *question answering*
- chatbot dan asisten percakapan

B. Apa yang sebaiknya dipelajari dulu

Untuk jalur NLP dan LLM, urutan yang masuk akal adalah:

- pahami bahwa **teks adalah data**
- mulai dari **tokenisasi** dan *text preprocessing*
- coba **TF-IDF** dan klasifikasi teks sederhana
- lanjut ke **sentiment analysis**
- baru masuk ke **LLM** dan *prompt engineering*

Outline utama juga secara eksplisit menyebut **TF-IDF sebagai jembatan ke NLP modern**.

C. Apa itu LLM secara praktis

LLM atau *Large Language Model* adalah model bahasa besar yang dilatih pada data teks skala besar dan mampu:

- memahami instruksi
- menjawab pertanyaan
- merangkum
- menghasilkan teks
- membantu menulis kode

Lampiran menjelaskan bahwa model-model modern seperti BERT, T5, PEGASUS, dan keluarga transformer telah mengubah NLP dengan kemampuan memahami konteks lebih dalam.

D. Tantangan penting di NLP/LLM

Lampiran juga mengingatkan bahwa NLP modern punya tantangan besar:

- ambiguitas bahasa
- bias data
- interpretabilitas
- etika
- bahasa sumber daya rendah
- informasi sensitif

Itu artinya, belajar NLP/LLM bukan hanya soal “menggunakan model besar”, tetapi juga memahami batasannya.

29.4 AI coding assistant

A. Mengapa ini sangat relevan

Outline utama buku menekankan bahwa **AI-assisted coding** dan *prompt engineering* layak dimasukkan secara ringan karena sudah menjadi keterampilan nyata dalam pengembangan Python modern. Ia juga memberi catatan penting bahwa AI coding assistant harus menjadi **partner belajar, bukan pengganti berpikir**.

Lampiran *Coding with AI* mendukung ini dengan sangat kuat. Di sana dijelaskan berbagai teknik memakai AI untuk:

- merancang arsitektur
- menjelaskan error
- memecah tugas coding menjadi langkah kecil
- menghasilkan tes
- membuat dokumentasi
- membandingkan pendekatan teknis
- membangun UI

- mempercepat debugging dan iterasi kode

B. Cara memakai AI coding assistant dengan sehat

Gunakan AI coding assistant untuk:

- menjelaskan error message
- memberi contoh kode awal
- membuat *boilerplate*
- menyusun test case
- membandingkan opsi desain
- merapikan dokumentasi

Tetapi tetap lakukan:

- verifikasi kode
- cek logika bisnis
- jalankan pengujian
- pahami apa yang diketik AI

Lampiran Morgan secara eksplisit menekankan bahwa hasil terbaik datang dari **menggabungkan kode yang dihasilkan AI dengan pengetahuan domain manusia**.

C. Keterampilan penting: menulis prompt yang baik

Lampiran juga menjelaskan teknik prompt untuk programmer, misalnya:

- tentukan pola arsitektur sejak awal
- minta output terstruktur
- sertakan error lengkap saat debugging
- pecah tugas rumit menjadi langkah-langkah kecil
- minta edge cases saat membuat test

Ini artinya, belajar AI coding assistant bukan hanya “menggunakan tool”, tetapi juga belajar **memberi instruksi teknis yang jelas**.

29.5 Deployment sederhana

A. Kenapa deployment penting

Outline utama secara eksplisit menyebut **deployment ringan** sebagai langkah lanjutan yang layak, dengan contoh:

- simpan model dengan `pickle` / `joblib`
- buat demo sederhana dengan **Streamlit** atau **Gradio**

Ini penting karena deployment membuat proyek Anda berubah dari:

- “notebook yang hanya berjalan di laptop”

menjadi:

- “alat kecil yang bisa dicoba orang lain”

B. Bentuk deployment paling cocok untuk pemula

Untuk pemula, deployment yang paling masuk akal adalah:

- **simpan model** ke file `.pkl`
- buat **demo prediksi sederhana**
- gunakan **Streamlit** atau **Gradio**

Ini cukup ringan, tidak menuntut infrastruktur berat, dan sangat cocok untuk portofolio awal.

C. Kenapa Streamlit atau Gradio

Karena keduanya membantu Anda membuat:

- form input
- tombol prediksi
- output hasil model

tanpa harus terlalu dalam ke *frontend*.

D. Langkah minimal deployment

Alur paling sederhana:

1. latih model
2. simpan dengan `joblib`
3. buat app kecil
4. muat model saat aplikasi dijalankan
5. tampilkan input dan prediksi

Ini sudah cukup kuat untuk menunjukkan bahwa Anda bisa membawa model ke bentuk yang lebih nyata.

29.6 Jalan belajar yang sehat: dari ML ke AI systems

Outline utama juga menyinggung bahwa setelah ML dasar, pembaca bisa mulai melihat horizon lain seperti:

- **transfer learning**
- **AI agents**
- **reinforcement learning**
- **autonomous systems**
- **etika AI: bias, privasi, transparansi**

Tetapi catatannya juga sangat penting: untuk buku pemula, topik-topik ini sebaiknya diposisikan sebagai **arah lanjutan**, bukan materi inti. Itu saran yang sangat sehat.

Artinya, setelah buku ini, Anda **tidak perlu belajar semuanya sekaligus**. Pilih satu jalur utama dulu.

29.7 Roadmap belajar 6 bulan

Bagian ini sangat penting karena outline utama memang secara eksplisit meminta **roadmap belajar 6 bulan setelah buku selesai**.

Berikut roadmap yang saya sarankan, realistis untuk pembaca yang sudah menamatkan buku ini dan mau lanjut secara konsisten.

Bulan 1 — Menguatkan fondasi dengan proyek ulang

Fokus:

- ulangi 2–3 mini project dari buku ini
- rapikan notebook
- buat repo GitHub
- pastikan bisa menjelaskan:
    - cleaning
    - feature
    - modeling
    - evaluasi
    - insight

Target:

- punya **minimal 2 proyek portofolio rapi**

Bulan 2 — Pendalaman supervised learning

Fokus:

- ulangi regresi dan klasifikasi dengan dataset baru
- bandingkan beberapa model
- latihan membaca confusion matrix, RMSE, R^2, precision, recall, F1
- mulai biasakan *cross-validation*

Target:

- nyaman membandingkan model, bukan hanya menjalankan satu model

Bulan 3 — Pendalaman unsupervised learning + visualisasi

Fokus:

- ulangi clustering
- coba PCA untuk visualisasi
- buat satu proyek segmentasi pelanggan yang lebih matang
- latihan menulis interpretasi bisnis

Target:

- punya **1 proyek clustering / exploratory insight** yang layak portofolio

Bulan 4 — Masuk ke deep learning dasar

Fokus:

- pahami neuron, layer, aktivasi, loss
- coba MLP sederhana
- belajar PyTorch atau Keras dasar
- jangan terlalu banyak teori; fokus ke intuisi dan eksperimen kecil

Target:

- membuat **1 notebook deep learning dasar**

Bulan 5 — Masuk ke NLP dan LLM

Fokus:

- tokenisasi dan preprocessing teks
- TF-IDF
- klasifikasi teks sederhana
- pengantar sentiment analysis
- coba LLM untuk tugas kecil
- mulai latihan prompt yang baik

Target:

- punya **1 proyek NLP sederhana**

Bulan 6 — Deployment ringan + AI coding assistant

Fokus:

- pilih proyek terbaik
- simpan model
- buat demo sederhana dengan Streamlit/Gradio
- gunakan AI coding assistant untuk mempercepat dokumentasi, debugging, dan UI ringan
- rapikan 3 proyek utama di GitHub

Target:

- punya **3 proyek portofolio kuat**
- minimal **1 demo sederhana** yang bisa dicoba

29.8 Cara memilih jalur spesialisasi setelah 6 bulan

Setelah roadmap awal ini, Anda bisa mulai memilih jalur yang lebih spesifik.

Jika suka angka, tabel, dan prediksi bisnis

Pilih fokus:

- data analysis
- machine learning tabular
- evaluasi model
- deployment sederhana

Jika suka teks dan bahasa

Pilih fokus:

- NLP
- LLM
- chatbot
- summarization
- text classification

Jika suka gambar, audio, atau sinyal

Pilih fokus:

- deep learning
- computer vision
- audio classification
- transfer learning

Jika suka membangun tool

Pilih fokus:

- AI coding assistant workflow
- MLOps dasar
- deployment
- aplikasi Streamlit/Gradio/Flask

Jika suka sistem yang lebih otonom

Pilih fokus:

- AI agents
- reinforcement learning
- autonomous systems

Tetapi sekali lagi, pilih **satu jalur utama lebih dulu**.

29.9 Apa yang sebaiknya tetap dijaga saat belajar lanjutan

Semakin maju topiknya, semakin penting menjaga kebiasaan dasar yang baik. Lampiran tentang versioning dan workflow modern menekankan pentingnya:

- reproducibility
- versioning
- documentation
- metadata
- evaluasi yang konsisten

Karena itu, apa pun jalur lanjutannya, tetap jaga kebiasaan berikut:

- simpan proyek dengan rapi
- dokumentasikan eksperimen
- bandingkan model secara jujur
- jangan langsung percaya output AI
- verifikasi hasil
- pahami data sebelum model
- pikirkan etika dan bias

Ini akan membuat pembelajaran Anda jauh lebih tahan lama.

29.10 Sumber latihan dan komunitas

Lampiran Jain juga menyarankan agar pembelajar terus bertumbuh lewat:

- proyek latihan
- dataset publik
- komunitas data science
- GitHub
- Kaggle
- Stack Overflow
- forum dan konferensi

Ini penting karena setelah buku selesai, belajar tidak boleh berhenti. Salah satu cara paling sehat adalah:

- ambil dataset baru
- buat proyek kecil
- unggah ke GitHub
- diskusikan dengan komunitas
- perbaiki proyek dari umpan balik

29.11 Kesalahan umum setelah selesai belajar dasar

Ada beberapa jebakan yang sangat sering terjadi setelah menamatkan materi dasar.

- **langsung ingin belajar semuanya sekaligus**
- **terlalu cepat lompat ke deep learning tanpa fondasi kuat**
- **terlalu percaya hasil AI coding assistant**
- **tidak menyiapkan portofolio**
- **belajar topik baru terus, tetapi tidak menyelesaikan proyek**
- **mengabaikan etika, bias, dan kualitas data**
- **tidak mendokumentasikan eksperimen**

Sikap yang lebih sehat adalah:

> **selesaikan proyek kecil, rapikan, dokumentasikan, lalu naik satu tingkat.**

29.12 Refleksi: setelah buku ini, yang paling penting adalah ritme

Bab ini seharusnya membuat pembaca tenang. Langkah berikutnya setelah buku ini **bukan perlombaan** untuk menguasai semua topik AI modern sekaligus. Justru yang paling penting adalah:

- ritme belajar yang konsisten
- proyek yang selesai
- pemahaman yang makin dalam
- dokumentasi yang makin rapi
- rasa ingin tahu yang tetap hidup

Outline utama buku juga sangat sehat dalam hal ini: bagian AI modern ditempatkan sebagai **arah lanjutan** dan **horizon berikutnya**, bukan beban teknis yang harus dipaksakan sekaligus.

Penutup Bab

Bab ini membahas **langkah berikutnya setelah buku ini** melalui lima arah utama: **deep learning dasar**, **NLP dan LLM**, **AI coding assistant**, **deployment sederhana**, dan **roadmap belajar 6 bulan**. Semua ini sangat selaras dengan outline utama buku yang memang menargetkan pembaca sampai cukup kuat di fondasi Python, data, dan ML klasik, lalu **mengenal arah lanjutan: deep learning, LLM, AI coding assistant, dan deployment ringan**, serta menyusun **roadmap belajar 6 bulan** yang realistis. Pembahasan ini juga kuat didukung oleh materi lampiran tentang evolusi NLP modern, penggunaan AI untuk coding, pentingnya dokumentasi dan versioning, serta pengembangan proyek dan komunitas belajar data science.

Pesan terpenting dari bab ini: setelah buku ini, Anda tidak perlu belajar semua hal sekaligus; yang paling penting adalah **memilih jalur yang masuk akal, membangun proyek yang selesai, dan terus naik tingkat secara konsisten**.

Tantangan kecil untuk pembaca: tulis satu rencana belajar pribadi 6 bulan versi Anda sendiri, pilih **satu jalur utama** (misalnya deep learning, NLP/LLM, atau deployment), lalu tentukan **3 proyek nyata** yang ingin Anda selesaikan sepanjang periode itu.

Bab 30. Penutup — Python sebagai Jalan Masuk ke Dunia AI

Pada akhirnya, pesan terpenting dari buku ini sangat sederhana: **Python bukan sekadar bahasa pemrograman, tetapi pintu masuk yang nyata, ramah, dan kuat ke dunia AI**. Melalui Python, pembaca tidak hanya belajar menulis kode, tetapi juga belajar memahami data, melihat pola, membangun model, membaca hasil, dan menyiapkan diri untuk melangkah ke wilayah AI yang lebih luas. Karena itu, Python layak dipandang bukan sebagai tujuan akhir, melainkan sebagai *foundation* yang membuka banyak kemungkinan.

Hal yang paling penting untuk diingat adalah bahwa perjalanan menuju AI tidak harus dilakukan dengan tergesa-gesa. **Belajar bertahap justru merupakan strategi yang paling sehat, paling realistis, dan paling kuat**. Dunia AI memang luas, penuh istilah, dan sering terlihat rumit dari kejauhan. Namun ketika dipelajari secara perlahan—mulai dari Python dasar, pengolahan data, statistik praktis, *machine learning* sederhana, hingga proyek kecil—semuanya menjadi jauh lebih masuk akal. Kemajuan yang kokoh hampir selalu lahir dari langkah-langkah kecil yang dijaga dengan sabar.

Buku ini juga ingin menegaskan bahwa **Anda tidak harus menjadi jenius matematika untuk mulai belajar AI**. Yang dibutuhkan di tahap awal bukanlah kesempurnaan teori, melainkan keberanian untuk memahami konsep secara bertahap, mencoba contoh nyata, membaca hasil dengan jernih, dan terus memperkuat intuisi. Matematika tentu akan sangat membantu seiring perjalanan menjadi semakin dalam, tetapi untuk memulai, yang paling dibutuhkan adalah rasa ingin tahu, kemauan belajar, dan kesediaan untuk terus berlatih.

Karena itu, jika ada satu sikap yang paling menentukan kemajuan, jawabannya bukan bakat bawaan, melainkan **konsistensi**. Kemampuan tumbuh bukan karena belajar sekali dalam ledakan semangat, melainkan karena terus kembali, terus berlatih, terus memperbaiki, dan terus menjaga ritme. Bahkan kemajuan kecil yang dilakukan berulang akan jauh lebih kuat daripada ambisi besar yang cepat padam. Dalam dunia Python dan AI, orang yang terus bergerak pelan tetapi stabil sering melangkah lebih jauh daripada orang yang ingin langsung mahir dalam waktu singkat.

Selain konsisten, **praktik nyata** adalah pembeda yang paling terasa. Membaca saja tidak cukup. Pemahaman menjadi kuat ketika pembaca benar-benar memuat data, membersihkan kesalahan, membuat visualisasi, melatih model, membaca metrik, lalu menuliskan insight. Dari praktik itulah rasa percaya diri tumbuh. Dari praktik pula, konsep yang tadinya terasa abstrak menjadi sesuatu yang bisa disentuh, diuji, dan diperbaiki. AI tidak lahir dari teori yang hanya disimpan di kepala, tetapi dari keberanian untuk berkotor tangan dengan data, kode, dan hasil yang belum tentu sempurna.

Di saat yang sama, **refleksi** juga tidak kalah penting. Belajar yang baik bukan hanya soal terus maju, tetapi juga soal berhenti sejenak untuk melihat apa yang sudah dipahami, apa yang masih kabur, bagian mana yang paling menarik, dan arah mana yang ingin diperdalam. Refleksi membuat proses belajar menjadi lebih sadar, lebih tenang, dan lebih sesuai dengan ritme masing-masing. Tidak semua orang akan jatuh cinta pada bagian yang sama. Ada yang lebih menikmati analisis data, ada yang tertarik pada modeling, ada yang bersemangat di NLP, ada yang justru menemukan daya tarik di clustering atau deployment. Semua itu wajar.

Yang juga sangat penting, **terus mencoba** harus menjadi bagian dari sikap belajar. Error, hasil model yang belum bagus, notebook yang berantakan, library yang bentrok, atau grafik yang aneh bukan tanda bahwa Anda gagal. Itu justru bagian normal dari perjalanan. Dunia Python dan AI bukan dunia yang menuntut Anda langsung sempurna, tetapi dunia yang menghargai orang yang mau mencoba lagi, memperbaiki lagi, dan belajar dari hasil yang belum sesuai harapan. Sering kali, kemajuan terbesar lahir bukan dari satu percobaan yang mulus, tetapi dari keberanian untuk tidak berhenti setelah percobaan yang pertama gagal.

Jika setelah membaca buku ini Anda masih merasa belum hebat, belum cepat, atau belum percaya diri, itu bukan masalah. **Rasa belum sempurna adalah bagian normal dari proses bertumbuh**. Yang jauh lebih penting adalah menyadari bahwa Anda sekarang sudah berada di tempat yang berbeda dibanding saat pertama mulai. Anda mungkin sudah lebih berani membuka terminal, lebih tenang membaca dataset, lebih paham arti korelasi, lebih mengenal evaluasi model, atau lebih siap menyusun notebook dan portofolio. Perubahan seperti ini sering tumbuh pelan-pelan, tetapi nilainya sangat besar.

Pada akhirnya, Python memberi sesuatu yang sangat berharga bagi pembelajar AI pemula: **akses**. Ia membuat dunia yang semula terlihat jauh dan rumit menjadi sesuatu yang bisa dipelajari, dicoba, dan dikuasai sedikit demi sedikit. Namun akses itu baru berubah menjadi kemampuan jika disertai oleh **belajar bertahap, konsistensi, praktik, refleksi, dan keberanian untuk terus mencoba**. Itulah inti perjalanan buku ini, dan itulah bekal yang paling penting untuk melangkah ke tahap berikutnya.

Pesan terakhir dari buku ini sederhana namun kuat:

> **Anda tidak perlu menunggu menjadi sempurna untuk mulai. Mulailah dengan apa yang ada, pelajari dengan sabar, praktikkan dengan sungguh-sungguh, dan biarkan kemampuan tumbuh seiring waktu. Python memberi pintu masuk ke dunia AI; selebihnya adalah keberanian Anda untuk terus melangkah.**

Lampiran A. Cheat Sheet Perintah Ubuntu 26.04 untuk Python

Lampiran ini dirancang sebagai **lembar cepat pakai** untuk pembaca yang ingin langsung bekerja di Ubuntu 26.04 tanpa harus mengingat semua perintah dari nol. Fokusnya bukan membahas teori sistem operasi secara panjang, tetapi memberi **perintah inti** yang paling sering dipakai saat belajar Python, data science, dan machine learning. Simpan lampiran ini dekat dengan laptop Anda, karena pada tahap awal belajar, kecepatan mengingat perintah dasar sangat membantu membangun ritme praktik.

A.1 Update sistem dan instal Python

Pada Ubuntu 26.04, Python biasanya sudah tersedia, tetapi tetap baik untuk memastikan sistem diperbarui dan paket inti terpasang.

```
sudo apt update
sudo apt upgrade -y
sudo apt install -y python3 python3-pip python3-venv
```

Keterangan singkat:

- `python3` untuk interpreter Python
- `python3-pip` untuk instal paket Python
- `python3-venv` untuk membuat *virtual environment*

A.2 Cek versi Python dan pip

Gunakan perintah berikut untuk memastikan Python dan pip sudah tersedia.

```
python3 --version
pip3 --version
```

Contoh keluaran:

```
Python 3.13.x
pip 25.x from /usr/lib/python3/dist-packages/pip (python
3.13)
```

A.3 Membuat folder proyek

Biasakan membuat folder proyek yang rapi sejak awal.

```
mkdir proyek-python
cd proyek-python
```

Untuk struktur yang lebih rapi:

```
mkdir -p proyek-python/{data,notebooks,src,outputs}
cd proyek-python
```

A.4 Membuat *virtual environment*

Sangat disarankan agar setiap proyek punya lingkungan Python sendiri.

```
python3 -m venv .venv
```

Nama `.venv` umum dipakai agar mudah dikenali.

A.5 Mengaktifkan *virtual environment*

```
source .venv/bin/activate
```

Jika aktif, biasanya prompt terminal akan berubah, misalnya:

```
(.venv) user@ubuntu:~/proyek-python$
```

A.6 Menonaktifkan *virtual environment*

Jika ingin keluar dari lingkungan virtual:

```
deactivate
```

A.7 Menginstal package Python

Contoh instal paket inti data science dan machine learning:

```
pip install numpy pandas matplotlib scikit-learn jupyter
```

Kalau ingin memperbarui pip lebih dulu:

```
pip install --upgrade pip
```

A.8 Menyimpan daftar package

Setelah proyek berjalan, simpan daftar package agar mudah direplikasi.

```
pip freeze > requirements.txt
```

Untuk instal ulang package dari file:

```
pip install -r requirements.txt
```

A.9 Menjalankan Python interaktif

Untuk mencoba perintah Python cepat:

```
python3
```

Keluar dari mode interaktif:

```
exit()
```

A.10 Menjalankan file Python

Misalnya file bernama `main.py`:

```
python3 main.py
```

Jika sudah aktif di `.venv`, bisa juga:

```
python main.py
```

A.11 Menjalankan Jupyter Notebook

Pastikan `jupyter` sudah diinstal, lalu jalankan:

```
jupyter notebook
```

Atau jika ingin antarmuka yang lebih modern:

```
jupyter lab
```

Notebook biasanya terbuka di browser lokal.

A.12 Perintah yang sangat sering dipakai

```
pwd
ls
ls -la
cd nama_folder
mkdir nama_folder
rm nama_file
cp sumber tujuan
mv lama baru
cat nama_file
```

A.13 *Cheatsheet* singkat Ubuntu 26.04 untuk Python

```
# update sistem
sudo apt update && sudo apt upgrade -y

# install Python + pip + venv
sudo apt install -y python3 python3-pip python3-venv

# cek versi
python3 --version
pip3 --version

# buat proyek
mkdir proyek-python && cd proyek-python

# buat virtual environment
python3 -m venv .venv

# aktifkan virtual environment
source .venv/bin/activate

# install package
pip install numpy pandas matplotlib scikit-learn jupyter

# simpan dependency
pip freeze > requirements.txt

# jalankan file Python
python3 main.py

# jalankan Jupyter
jupyter notebook
```

Lampiran B. *Cheat Sheet* Python Dasar

Lampiran ini merangkum sintaks Python yang paling sering dipakai saat belajar AI dan machine learning. Fokusnya adalah **mengingat pola**, bukan menghafal semua kemungkinan variasi.

B.1 Tipe data dasar

```
nama = "Onno"
umur = 25
tinggi = 170.5
aktif = True
```

Jenis umum:

- `str` → teks
- `int` → bilangan bulat
- `float` → bilangan pecahan
- `bool` → `True` / `False`

Cek tipe data:

```
print(type(nama))
print(type(umur))
```

B.2 Struktur data penting

List

```
angka = [1, 2, 3, 4]
print(angka[0])
```

Tuple

```
titik = (10, 20)
```

Set

```
unik = {1, 2, 2, 3}
print(unik)
```

Dictionary

```
mahasiswa = {
    "nama": "Budi",
    "nilai": 85
}
print(mahasiswa["nama"])
```

B.3 Percabangan if

```
nilai = 80

if nilai >= 75:
    print("Lulus")
else:
    print("Tidak lulus")
```

B.4 Loop for

```
for i in range(5):
    print(i)
```

Loop untuk list:

```
buah = ["apel", "jeruk", "mangga"]

for item in buah:
    print(item)
```

B.5 Loop while

```
x = 0

while x < 5:
    print(x)
    x += 1
```

B.6 Fungsi

```
def sapa(nama):
    return f"Halo, {nama}"

hasil = sapa("Onno")
print(hasil)
```

Fungsi dengan dua parameter:

```
def tambah(a, b):
    return a + b

print(tambah(3, 5))
```

B.7 Input pengguna

```
nama = input("Masukkan nama: ")
print("Halo,", nama)
```

B.8 Membaca file teks

```
with open("catatan.txt", "r") as file:
    isi = file.read()
```

```
print(isi)
```

B.9 Menulis file teks

```
with open("hasil.txt", "w") as file:
    file.write("Belajar Python itu menyenangkan")
```

B.10 Membaca CSV dengan modul standar

```
import csv

with open("data.csv", "r") as file:
    reader = csv.reader(file)
    for row in reader:
        print(row)
```

B.11 Membaca JSON

```
import json

with open("data.json", "r") as file:
    data = json.load(file)

print(data)
```

B.12 Penanganan error dasar

```
try:
    angka = int(input("Masukkan angka: "))
    print(angka)
except ValueError:
    print("Input harus angka")
```

B.13 *Cheat Sheet* super singkat Python dasar

```
# variabel
x = 10
nama = "Andi"

# list
angka = [1, 2, 3]

# if
if x > 5:
    print("Besar")

# for
for i in range(3):
    print(i)

# fungsi
def kali(a, b):
    return a * b

# file
with open("file.txt", "r") as f:
    isi = f.read()
```

Lampiran C. Cheat Sheet Data Science

Lampiran ini merangkum fungsi inti dari empat library yang sering dipakai dalam buku ini:

- **NumPy**
- **Pandas**
- **Matplotlib**
- **scikit-learn**

C.1 NumPy

NumPy dipakai untuk komputasi numerik berbasis array.

Import

```
import numpy as np
```

Membuat array

```
a = np.array([1, 2, 3, 4])
print(a)
```

Array 2 dimensi

```
m = np.array([[1, 2], [3, 4]])
print(m)
```

Bentuk array

```
print(a.shape)
print(m.shape)
```

Membuat array khusus

```
np.zeros((2, 3))
np.ones((2, 2))
np.arange(0, 10, 2)
np.linspace(0, 1, 5)
```

Statistik dasar

```
print(np.mean(a))
print(np.median(a))
print(np.std(a))
print(np.min(a))
print(np.max(a))
```

Reshape

```
x = np.arange(6)
y = x.reshape(2, 3)
print(y)
```

C.2 Pandas

Pandas dipakai untuk bekerja dengan tabel data.

Import

```
import pandas as pd
```

Membaca CSV

```
df = pd.read_csv("data.csv")
```

Lima baris pertama

```
print(df.head())
```

Info data

```
print(df.info())
```

Statistik ringkas

```
print(df.describe())
```

Pilih kolom

```
print(df["nama_kolom"])
```

Pilih beberapa kolom

```
print(df[["kolom1", "kolom2"]])
```

Filter data

```
print(df[df["nilai"] > 80])
```

Urutkan data

```
print(df.sort_values(by="nilai", ascending=False))
```

Groupby

```
print(df.groupby("kelas")["nilai"].mean())
```

Missing value

```
print(df.isnull().sum())
df["nilai"] = df["nilai"].fillna(df["nilai"].median())
```

Simpan CSV

```
df.to_csv("hasil.csv", index=False)
```

C.3 Matplotlib

Matplotlib dipakai untuk visualisasi data.

Import

```
import matplotlib.pyplot as plt
```

Line plot

```
x = [1, 2, 3, 4]
y = [10, 20, 15, 25]

plt.plot(x, y)
plt.title("Line Plot")
plt.xlabel("X")
plt.ylabel("Y")
plt.show()
```

Bar plot

```
kategori = ["A", "B", "C"]
nilai = [5, 7, 3]

plt.bar(kategori, nilai)
plt.show()
```

Histogram

```
data = [1, 2, 2, 3, 3, 3, 4]
plt.hist(data)
plt.show()
```

Scatter plot

```
x = [1, 2, 3, 4]
y = [2, 4, 5, 4]

plt.scatter(x, y)
plt.show()
```

C.4 scikit-learn

scikit-learn dipakai untuk machine learning klasik.

Import umum

```
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LinearRegression,
LogisticRegression
from sklearn.cluster import KMeans
from sklearn.metrics import mean_squared_error,
accuracy_score
```

Train-test split

```
X_train, X_test, y_train, y_test = train_test_split(
    X, y, test_size=0.2, random_state=42
)
```

Linear Regression

```
model = LinearRegression()
model.fit(X_train, y_train)
y_pred = model.predict(X_test)
```

Logistic Regression

```
model = LogisticRegression()
model.fit(X_train, y_train)
y_pred = model.predict(X_test)
```

KMeans

```
model = KMeans(n_clusters=3, random_state=42, n_init=10)
labels = model.fit_predict(X)
```

Scaling

```
from sklearn.preprocessing import StandardScaler

scaler = StandardScaler()
X_scaled = scaler.fit_transform(X)
```

Lampiran D. Template Code Machine Learning

Lampiran ini berisi template yang bisa langsung dipakai dan dimodifikasi untuk tiga jenis tugas utama:

- **regresi**
- **klasifikasi**
- **clustering**

D.1 Template regresi

```
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LinearRegression
from sklearn.metrics import mean_absolute_error, mean_squared_error,
r2_score
import math

# 1. Load data
df = pd.read_csv("data.csv")

# 2. Pilih fitur dan target
X = df[["fitur1", "fitur2", "fitur3"]]
y = df["target"]

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

# 4. Train model
model = LinearRegression()
model.fit(X_train, y_train)

# 5. Prediksi
y_pred = model.predict(X_test)

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

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

D.2 Template klasifikasi

```
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import accuracy_score, precision_score,
recall_score, f1_score, confusion_matrix

# 1. Load data
df = pd.read_csv("data.csv")

# 2. Pilih fitur dan target
X = df[["fitur1", "fitur2", "fitur3"]]
y = df["target"]

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

# 4. Scaling
scaler = StandardScaler()
X_train_scaled = scaler.fit_transform(X_train)
X_test_scaled = scaler.transform(X_test)

# 5. Train model
model = LogisticRegression()
model.fit(X_train_scaled, y_train)

# 6. Prediksi
y_pred = model.predict(X_test_scaled)

# 7. Evaluasi
print("Accuracy :", accuracy_score(y_test, y_pred))
print("Precision:", precision_score(y_test, y_pred))
print("Recall   :", recall_score(y_test, y_pred))
print("F1-score :", f1_score(y_test, y_pred))
print("Confusion Matrix:")
print(confusion_matrix(y_test, y_pred))
```

D.3 Template clustering

```
import pandas as pd
from sklearn.preprocessing import StandardScaler
from sklearn.cluster import KMeans

# 1. Load data
df = pd.read_csv("data.csv")

# 2. Pilih fitur
X = df[["fitur1", "fitur2", "fitur3"]]

# 3. Scaling
scaler = StandardScaler()
X_scaled = scaler.fit_transform(X)

# 4. Clustering
model = KMeans(n_clusters=3, random_state=42, n_init=10)
df["cluster"] = model.fit_predict(X_scaled)

# 5. Lihat hasil
print(df.head())
print(df.groupby("cluster").mean(numeric_only=True))
```

Lampiran E. Dataset Gratis untuk Latihan

Lampiran ini berisi daftar dataset gratis yang sangat cocok untuk latihan Python, data science, dan machine learning dasar. Tujuannya adalah memberi pembaca titik mulai yang mudah, terkenal, dan cukup aman dipakai untuk eksplorasi.

E.1 Iris

Jenis masalah: klasifikasi
Cocok untuk: Logistic Regression, KNN, Decision Tree, Random Forest
Tentang dataset: berisi ukuran bunga iris dari beberapa spesies.

Contoh akses via scikit-learn:

```
from sklearn.datasets import load_iris

data = load_iris()
X = data.data
y = data.target
print(data.feature_names)
print(data.target_names)
```

E.2 Titanic

Jenis masalah: klasifikasi
Cocok untuk: prediksi selamat / tidak selamat
Tentang dataset: data penumpang Titanic dengan fitur seperti usia, jenis kelamin, kelas, dan status selamat.

Biasanya tersedia di Kaggle atau berbagai repository pembelajaran.

E.3 Wine

Jenis masalah: klasifikasi
Cocok untuk: klasifikasi beberapa kelas
Tentang dataset: fitur kimia dari beberapa jenis wine.

Contoh akses:

```
from sklearn.datasets import load_wine

data = load_wine()
X = data.data
y = data.target
print(data.feature_names)
```

E.4 Housing

Jenis masalah: regresi
Cocok untuk: prediksi harga rumah
Tentang dataset: sangat cocok untuk latihan regresi, evaluasi model, dan feature importance.

Contoh dataset yang mudah dipakai:

- California Housing dari scikit-learn

```
from sklearn.datasets import fetch_california_housing

data = fetch_california_housing(as_frame=True)
df = data.frame
print(df.head())
```

E.5 Customer segmentation

Jenis masalah: clustering
Cocok untuk: K-Means, PCA, visualisasi cluster
Tentang dataset: biasanya berisi fitur seperti pendapatan tahunan, skor belanja, dan frekuensi transaksi.

Sangat cocok untuk mini project *unsupervised learning*.

E.6 Dataset synthetic

Jenis masalah: regresi, klasifikasi, clustering
Cocok untuk: latihan cepat tanpa harus mencari dataset riil
Tentang dataset: dibuat secara artifisial untuk memahami perilaku model.

Contoh:

Synthetic regression

```
from sklearn.datasets import make_regression

X, y = make_regression(
    n_samples=100,
    n_features=2,
    noise=10,
    random_state=42
)
```

Synthetic classification

```
from sklearn.datasets import make_classification

X, y = make_classification(
    n_samples=100,
```

```
    n_features=4,
    n_informative=2,
    n_redundant=0,
    random_state=42
)
```

Synthetic clustering

```
from sklearn.datasets import make_blobs

X, y = make_blobs(
    n_samples=100,
    centers=3,
    n_features=2,
    random_state=42
)
```

Referensi

Abdullah, M. (2025). *#1 book for Python programming: Learn from basic programming to AI and automation*. Minhaj Abdullah.

Jain, A. (2025). *Python for data analysis: A beginner's guide to data science*. Aniket Jain.

Knox, S. W. (2026). *Machine learning: A concise introduction* (2nd ed.). John Wiley & Sons, Inc.

Lim, K. G. (2026). *Machine learning in business finance using Python*. World Scientific Publishing Co. Pte. Ltd.

Malato, G. (2025). *Statistics with Python: 100 solved exercises for data analysis*. Gianluca Malato.

Morgan, J. C. (2025). *Coding with AI: Examples in Python*. Manning Publications.

Purbo, O. W. (2026). *Python untuk pemula menuju AI & machine learning* [Unpublished manuscript].

Saltz, J. S., & Stanton, J. M. (2025). *An introduction to data science with Python*. Sage.

Uddin, M. I., Aziz, F., & Jafari, H. (Eds.). (2026). *Computational intelligent systems: Applications of AI and machine learning*. Springer.

Van Der Post, H. (2023). *Python fusion: Integrating Excel mastery with financial modeling*. Reactive Publishing.

Van Der Post, H. (2024). *AI in FP&A: Quantitative FP&A: The ultimate guide to financial planning & analysis with Python*. Reactive Publishing.

Van Der Post, H. (2025a). *AI agents with Python: Build autonomous systems that think, learn, and act*. Reactive Publishing.

Van Der Post, H. (2025b). *AI with Python: Practical applications and advanced techniques*. Reactive Publishing.

Tentang Penulis



Onno W. Purbo, saat ini bertugas sebagai rektor di Institut Teknologi Tangerang Selatan (ITTS). Onno memperoleh gelar Ph.D bidang Electrical Engineering dari University of Waterloo, Canada, adalah seorang copyleft, educator dan ICT evangelist. Dia sudah mempublikasikan 50++ buku, termasuk free ICT ebook untuk sekolah tahun 2008. Beberapa buku terakhirnya adalah "Internet-TCP/IP: Konsep Dan Implementasi", 2018; "Sistem Operasi, Konsep Dan Membuat Linux OpenWRT Dan ROM Android", 2019; "IPv6 Untuk Mendukung Operasi Jaringan Dan Domain Name System", 2019; "Kubernetes untuk Pemula", 2024 dan "Membuat Operator Seluler 5G Sendiri", 2024.

Dia memimpin sambungan pertama Internet di Institut Teknologi Bandung, tahun 1993-2000, dan menggunakannya untuk membuat jaringan Internet pendidikan yang pertama di Indonesia. Dia membebaskan frekuensi WiFi, memperkenalkan RT/RW-net, antenna Wajanbolic dan jaringan selular OpenBTS dan private 5G sendiri. Dia memimpin jaringan telepon pertama di atas Internet, VoIP Merdeka, yang kemudian hari dikenal sebagai VoIP Rakyat berbasis SIP dan menggunakan kode area +62520 dan +62521. Dia saat ini aktif memperkenalkan e-Learning, dan menjalankan server e-Learning di http://lms.onnocenter.or.id/moodle/ 66,000++ siswa / mahasiswa dan https://opencourse.itts.ac.id 39.000+ mahasiswa secara gratis. Email: onno@indo.net.id