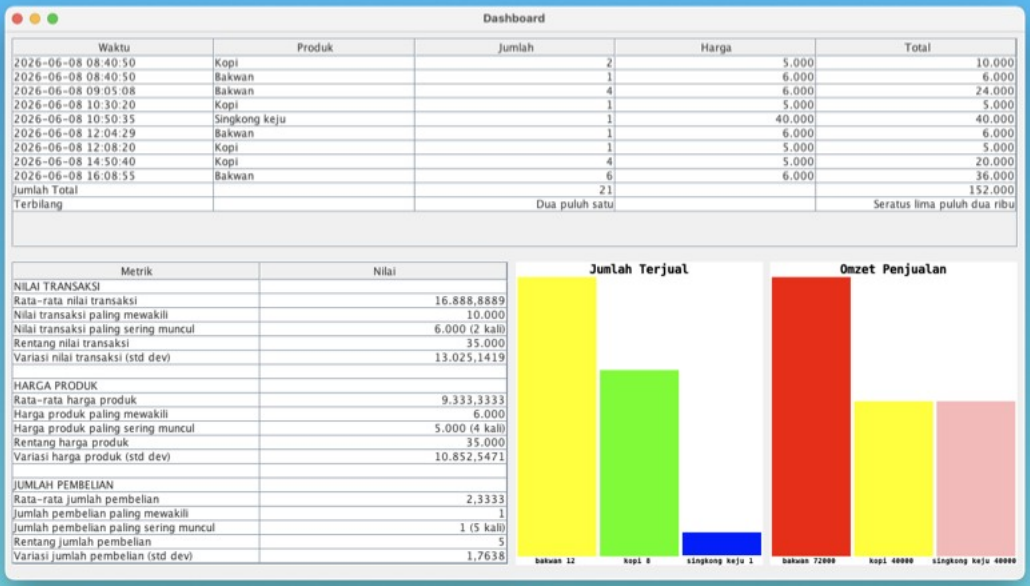
Contoh Analisis Data Sederhana

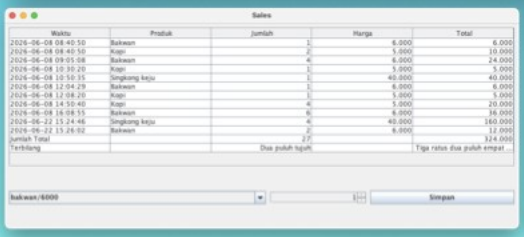
dengan

Bahasa Singkong



Waktu	Produk	Jumlah	Harga	Total
2026-06-08 08:40:50	Kopi	2	5.000	10.000
2026-06-08 08:40:50	Bakwan	1	6.000	6.000
2026-06-08 09:05:08	Bakwan	4	6.000	24.000
2026-06-08 10:30:20	Kopi	1	5.000	5.000
2026-06-08 10:50:35	Singkong keju	1	40.000	40.000
2026-06-08 12:04:29	Bakwan	1	6.000	6.000
2026-06-08 12:08:20	Kopi	1	5.000	5.000
2026-06-08 14:50:40	Kopi	4	5.000	20.000
2026-06-08 16:08:55	Bakwan	6	6.000	36.000
Jumlah Total		21		152.000
Terbilang		Dua puluh satu		Seratus lima puluh dua ribu

Metrik	Nilai
NILAI TRANSAKSI	
Rata-rata nilai transaksi	16.888,8889
Nilai transaksi paling mewakili	10.000
Nilai transaksi paling sering muncul	6.000 (2 kali)
Rentang nilai transaksi	35.000
Variasi nilai transaksi (std dev)	13.025,1419
HARGA PRODUK	
Rata-rata harga produk	9.333,3333
Harga produk paling mewakili	6.000
Harga produk paling sering muncul	5.000 (4 kali)
Rentang harga produk	35.000
Variasi harga produk (std dev)	10.852,5471
JUMLAH PEMBELIAN	
Rata-rata jumlah pembelian	2,3333
Jumlah pembelian paling mewakili	1
Jumlah pembelian paling sering muncul	1 (5 kali)
Rentang jumlah pembelian	5
Variasi jumlah pembelian (std dev)	1,7638



waktu	produk	jumlah	harga
2026-06-08 08:40:50	Kopi	2	5000
2026-06-08 08:40:50	bakwan	1	6000
2026-06-08 09:05:08	bakwan	4	Rp 6000
2026-06-08 10:30:20	kopi		5000
2026-06-08 10:50:35	Sinkong Keju	1	40000
08/06/2026 12:04:29	bakwan	1	6000
2026-06-08 12:08:20	kpi	1	5000
2026-06-08 14:50:40	kopi	4	5000
2026-06-08 16:08:55	Bak wan	6	6000
2026-06-08 16:20:58	bakwan	3	

Dr. Noprianto • Dr. Buyung Sofiarto Munir • Dr. Maria Seraphina Astriani

Penerbit: PT. Stabil Standar Sinergi

Contoh Analisis Data Sederhana Dengan Bahasa Singkong

Penulis: Dr. Noprianto, Dr. Buyung Sofiarto Munir, Dr. Maria Seraphina Astriani

ISBN: (diajukan pada 26 Juni 2026, perbaikan pada 9 Juli 2026).

Penerbit:
PT. Stabil Standar Sinergi

Alamat	Puri Indah Financial Tower Lantai 6, Unit 0612 Jl. Puri Lingkar Dalam Blok T8, Puri Indah, Kembangan, Jakarta Barat 11610
Website	www.singkong.dev
Email	info@singkong.dev

Dalam mempersiapkan dan ketika menulis buku ini, saya (Noprianto) berdiskusi dengan Gemini. Diskusi mencakup tata cara penulisan, tahapan dalam analisis data, sampai review kode program yang saya lampirkan (sekali, Gemini menemukan satu bug pada bagian membersihkan data).

Oleh karena itu, untuk bagian yang saya tulis dalam buku ini, teriring ucapan terima kasih, saya memberikan kredit bahwa buku ini “Ditulis dengan diskusi bersama Google Gemini.”

Daftar Isi

Kata Pengantar

Buku ini berisi sejumlah contoh dan penjelasan langkah demi langkah yang mudah dipahami untuk analisis data sederhana, dengan bahasa pemrograman Singkong.

Pembahasan diawali dengan contoh data penjualan yang tidak bersih, yang merepresentasikan kondisi yang umum terjadi, sebagai hasil dari sistem yang berjalan. Sejumlah contoh pembersihan data kemudian dibahas, sebelum pada akhirnya dilanjutkan pada analisis datanya.

Sebagai pelengkap, buku ini juga menyajikan dashboard GUI dengan chart, program entri data sederhana, contoh bekerja dengan sumber data lain (sistem database relasional dan web service), dan contoh laporan sederhana.

Jakarta, Juli 2026
Tim penulis

Untuk referensi dan dokumentasi lengkap bahasa Singkong, Anda mungkin ingin membaca buku-buku gratis berikut:

- Mengenal dan Menggunakan Bahasa Pemrograman Singkong (ISBN: 978-602-52770-1-6, Dr. Noprianto).
- Contoh dan Penjelasan Bahasa Singkong: Dasar-dasar Aplikasi GUI (ISBN: 978-602-52770-3-0, Dr. Noprianto, Dr. Karto Iskandar, Benfano Soewito, Ph.D.).
- Contoh dan Penjelasan Bahasa Singkong: Mahir Bekerja dengan GUI (ISBN: 978-602-52770-4-7, Dr. Noprianto, Dr. Wartika, Dr. Ford Lumban Gaol).
- Contoh dan Penjelasan Bahasa Singkong: Bekerja dengan Database Relasional (ISBN: 978-602-52770-5-4, Dr. Noprianto, Dr. Maria Seraphina Astriani, Dr. Fredy Purnomo).
- Contoh dan Penjelasan Bahasa Singkong: Aplikasi Web dan Topik Lanjutan (ISBN: 978-602-52770-6-1, Dr. Noprianto, Dr. Buyung Sofiarto Munir, Dr. Sarwo).
- Dasar-dasar Algoritma, Struktur Data, dan Pemrograman dengan Bahasa Singkong (ISBN: 978-602-52770-8-5, Dr. Noprianto, Budiman).

Semua buku tersebut juga dapat dibaca dengan mengunjungi: singkong.dev

Persiapan

Siapkanlah sebuah komputer, yang dilengkapi layar, keyboard, dan mouse/trackpad. Apabila Anda ingin menggunakan smartphone ataupun TV Android, tutorialnya tersedia di singkong.dev.

Untuk perangkat keras komputernya, dapat menggunakan spesifikasi komputer mulai dari yang terbaru ataupun yang telah dijual sekitar tahun 2000. Yang penting, dapat menjalankan salah satu dari daftar sistem operasi berikut.

Interpreter Singkong (dan program yang Anda buat nantinya) dapat berjalan pada berbagai sistem operasi berikut:

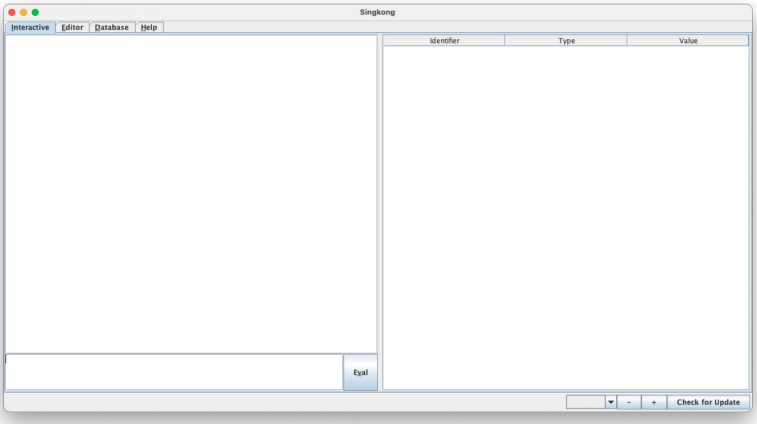
- macOS (mulai dari Mac OS X 10.4 Tiger)
- Windows (mulai dari Windows 98)
- Linux (mulai yang dirilis sejak awal 2000-an; juga termasuk Raspberry Pi OS, Termux di Android, Armbian, Debian di Android)
- Chrome OS (sejak tersedia Linux development environment, telah diuji pada versi 101 di chromebook)
- Solaris (diuji pada versi 11.4)
- FreeBSD (diuji pada versi 13.0 dan 12.1)
- OpenBSD (diuji pada versi 7.0 dan 6.6)
- NetBSD (diuji pada versi 9.2 dan 9.0)
- eComStation (diuji pada versi 2.1)
- ReactOS (diuji pada versi alpha)

Setelah komputer siap, lakukanlah instalasi Java, apabila belum terinstal sebelumnya. Secara teknis, Anda hanya membutuhkan Java Runtime Environment, versi 5.0 atau lebih baru. Versi 5.0 dirilis pada tahun 2004 (Sekitar 22 tahun lalu pada saat buku ini ditulis). Gunakan versi yang masih didukung secara teknis, apabila memungkinkan.

(Kenapa perlu menginstalasi Java? Karena interpreter Singkong ditulis dengan bahasa Java dan bahasa Singkong itu sendiri.)

Sebagai langkah terakhir, downloadlah interpreter Singkong, yang akan selalu didistribusikan sebagai file jar tunggal (Singkong.jar). Downloadlah selalu dari https://nopri.github.io/Singkong.jar. Pada saat buku ini ditulis, ukurannya hanya 4,4 MB dan berisikan semua yang diperlukan untuk mengikuti semua pembahasan dalam buku ini. Pastikanlah Anda menggunakan versi terbaru.

Apabila Singkong.jar telah dapat dijalankan (misal dengan klik pada filenya), maka persiapan sudah selesai. Perhatikanlah bahwa kita aktif pada tab Interactive untuk menguji kode program singkat secara langsung dan tab Editor untuk mengetikkan, menyimpan/membuka, dan menjalankan kode program yang lebih panjang (silahkan menggunakan editor pilihan Anda, apabila diperlukan).



Apabila langkah detil instalasi Java dan menjalankan Singkong.jar diperlukan, kita juga membahasnya misal di buku: Mengenal dan Menggunakan Bahasa Pemrograman Singkong.

Kode Sumber

Keseluruhan kode sumber (source code) yang digunakan dalam buku ini tersedia di https://github.com/nopri, repositori example, dan direktori data_analysis di dalamnya.

Di dalam buku ini pun, keseluruhan kode tersebut tersedia lengkap, hanya saja disajikan sebagian demi sebagian, dan langsung dibahas, dengan pola berikut:

- Disebutkan secara eksplisit, sebagai contoh: *Baris 48 sampai 56 berikut tidaklah panjang...*
- Diikuti dengan kodenya itu sendiri, diberikan border, dengan font monospace. Baris kosong, apabila dituliskan, menjadi bagian dari kode tersebut (agar lebih terbaca dan ikut dalam penanda nomor baris). Contoh:

```
var clean_qty = fn(s) {
    var s = replace(lower(trim(ifnull(s, ""))), " ", "")
    var a = extract(s, NUM)
    if (empty(a)) {
        var a = ["1"]
    }
    return number(a[0], 0, true)
}
```

- Tanpa disebutkan eksplisit barisnya atau tanpa border, dimaksudkan sebagai pembahasan bagian dari kode yang ditampilkan sebelumnya. Misal:

Ingatkah Anda dengan baris 8 ini?

```
var NUM = "(\d+)"
```

Kita akan gunakan dalam pemanggilan fungsi extract berikut:

```
  var a = extract(s, NUM)
```

- Terkadang, pembahasan dilanjutkan dengan cuplikan kode program yang ditujukan untuk dijalankan pada interactive evaluator Singkong, untuk mencontohkan lebih lanjut.

Artinya, ambil angka saja (\d+) dalam STRING yang dilewatkan. Perhatikanlah contoh berikut:

```
> var NUM = "(\d+)"
> var s = "Rp 6000"
> var a = extract(s, NUM)
> a
["6000"]
```

Data Penjualan yang Tidak Bersih

Anggaplah kita baru saja membuka warung makan dan berikut adalah keseluruhan data penjualan yang kita miliki.

waktu	produk	jumlah	harga
2026-06-08 08:40:50	Kopi	2	5000
2026-06-08 08:40:50	bakwan	1	6000
2026-06-08 09:05:08	bakwan	4	Rp 6000
2026-06-08 10:30:20	kopi		5000
2026-06-08 10:50:35	Sinkong Keju	1	40000
08/06/2026 12:04:29	bakwan	1	6000
2026-06-08 12:08:20	kpi	1	5000
2026-06-08 14:50:40	kopi	4	5000
2026-06-08 16:08:55	Bak wan	6	6000
2026-06-08 16:20:58	bakwan	3	

Apakah Anda menemukan masalah pada data penjualan tersebut? Tentunya yang dimaksud bukan pada seberapa banyak yang terjual atau harga jual setiap porsinya :)

Mari sama-sama kita temukan.

Data yang tidak konsisten

Yang paling terlihat tentunya pada kolom produk. Ada kopi yang salah ketik sebagai:

- Kopi, dengan K huruf besar (baris 1)
- kopi, dengan k huruf kecil (baris 4 dan 8)
- kpi, yang mana, maksudnya adalah kopi (baris 7)

Bagaimana dengan bakwan? Juga terdapat salah ketik, menjadi Bak wan (huruf besar pada B, spasi antara bak dan wan) di baris 9.

Ada lagi? Pastinya. Yaitu, Sinkong Keju (baris 5). Selain salah ketik pada kata singkong, terdapat sejumlah spasi yang tidak perlu antara singkong dan keju. Padahal, sebagaimana dapat kita temukan sehari-hari, singkong dan keju harusnya cukup akrab. Tidak perlu banyak-banyak spasi.

Salah ketik yang berulang-ulang ini berujung pada data yang tidak konsisten. Dan, ini artinya, produk yang sebenarnya sama dianggap sebagai produk berbeda. Pada contoh kopi sebelumnya, ini bisa saja dianggap sebagai tiga produk berbeda.

Katakanlah, kita akan menggunakan program spreadsheet untuk mengolah data tersebut. Tentunya, data yang salah ketik tersebut dapat mengacaukan hasil dari fungsi count, misalnya.

Oleh karena itu, kita perlu pastikan konsistensinya.

Nilai yang hilang

Di baris ke-4, terdapat penjualan kopi. Harga satuannya benar, tapi jumlahnya tidak diisikan. Bagaimana kita dapat mengartikan jumlah ini? Bisakah seseorang membeli nol cangkir kopi? Tentunya tidak mungkin. Karena, kalau memang tidak ada penjualan, tentunya tidak perlu kita tambahkan dalam data penjualan.

Batas minimal sebuah transaksi barang fisik adalah satu unit. Dan, dalam contoh ini, mengasumsikan nilai minimal tersebut cukuplah aman, sekedar untuk ikut dalam perhitungan. Tanpa membuat omzet menjadi bengkak.

Dalam hal ini, kita akan mengisi data kosong dengan tebakan yang logis. Kita melakukan imputasi data.

Bagaimana dengan bakwan di baris terakhir? Jumlahnya ada, tapi harga jualnya tidak diisi. Kasus ini berbeda dengan kopi yang tadi.

Ketika harga satuan tidak diisi, kita kehilangan komponen vital untuk melakukan perhitungan secara akurat, dimana total = jumlah x harga.

Kita harus tebak berapa? Apakah 6.000 seperti sebelumnya? Bagaimana kalau sedang diskon? Ataukah tidak diisi karena gratis?

Kali ini, kita tidak ingin menebak karena harga bersifat sensitif terhadap omzet dan laba rugi bisnis. Artinya, apabila kita memaksakan harga tertentu, maka laporan akhir akan menjadi tidak akurat.

Dalam buku ini, pendekatan kita adalah menghapus baris data tersebut. Nantinya, kita perlu mengevaluasi apakah ini memang hanya terjadi pada sebagian kecil data.

Format yang tidak konsisten

Yang paling mudah terlihat adalah pada baris ke-3, di mana harga satuan dituliskan sebagai Rp 6000. Hanya baris ini yang mencantumkan teks Rp. Untuk kebutuhan diproses oleh komputer, kita tidak membutuhkan teks tambahan tersebut.

Ketika input manual dilakukan, atau ketika data sebelumnya dikelola dengan spreadsheet, kita mungkin juga akan menemukan variasi lain seperti pemisah ribuan dengan karakter titik atau koma.

Di baris data tersebut, kita akan mengambil 6000 dari Rp 6000 tersebut. Hanya angkanya saja.

Ada lagi? Ya. Format tanggal pada baris ke-6 (dd/mm/yyyy) tidak sama dengan format pada baris-baris lainnya (yyyy-mm-dd). Masih untung, bagian waktunya dituliskan lengkap sampai detik, formatnya konsisten, dan nilainya valid. Sama seperti ketika input manual atau lewat program spreadsheet, variasi format ini bisa saja bermacam-macam.

Di baris data tersebut, kita dapat mencoba untuk mengartikan tanggal yang tertulis.

Tipe data

Bayangkanlah ketika data penjualan kita sudah bersih. Nama produk dan format sudah konsisten. Nilai yang

hilang sudah diperbaiki. Baris data yang tidak valid telah dihapus.

Apa yang kita miliki masih berupa data teks, bukan? Misal, 6000 teks tidaklah sama dengan 6000 bilangan, ketika perlu diproses oleh komputer. Akan tetapi, akan lebih mudah dikonversi apabila formatnya konsisten. Kita akan membahas ini di bab tersendiri, untuk membersihkan data.

Masalah lainnya

Untungnya, data kita tidak memiliki berbagai masalah berikut.

Data duplikat

Apabila data dicatat lewat program, kemudian kasir menekan tombol simpan lebih dari sekali untuk penjualan yang sama, dan program yang digunakan tidak mengantisipasi ini, maka dapat terjadi kondisi baris transaksi yang sama persis tersimpan lebih dari sekali. Pada sistem yang dicatat manual lewat tulisan tangan, kemungkinan ini tentunya lebih kecil (karena, butuh usaha ekstra untuk menulis dengan tangan, untuk transaksi yang sama persis).

Masalah duplikasi data ini dapat pula ditemukan pada hasil penggabungan data dari beberapa sumber.

Spasi berlebih/karakter tersembunyi

Pada baris data Sinkong Keju, spasi diantara singkong dan keju dapat lebih dicermati oleh kita, dibandingkan dengan “ bakwan” atau “bakwan “ misalnya, ketika spasi berlebih ditemukan sebelum atau setelah nama produk.

Atau, ketika data didapatkan lewat copy-paste dari program atau format dokumen tertentu. Bisa saja, karakter non-breaking space, tab, new line, atau kontrol pemformatan dapat ikut terkopi. Karakter-karakter tersebut ikut menyusup, namun mungkin tampak seperti spasi kosong di layar monitor. Masalahnya, ketika diproses, dapat mengacaukan logika validasi program kita.

Nilai diluar batas jangkauan

Bagaimana kalau jumlah barang dijual dituliskan sebagai bilangan negatif, misal -5? Bagaimana mungkin seseorang membeli -5 bakwan?

Atau, ketika waktu penjualan melompat ke masa depan yang belum terjadi.

Dengan hanya sepuluh baris data, dari satu hari penjualan, kita menemukan sejumlah masalah. Bagaimana dengan sistem yang telah berjalan tahunan, dan mungkin telah melewati sejumlah migrasi data, perubahan program yang digunakan, penyesuaian dengan kebijakan perusahaan, dan berbagai kemungkinan kesalahan input manual?

Di bab berikut, kita akan membahas dinamika sistem yang berjalan dan datanya.

Sistem Berjalan dan Data yang Dihasilkan

Dr. Buyung Sofiarto Munir

Sistem yang berjalan akan menghasilkan data. Idealnya, data yang dihasilkan siap untuk dianalisis sebagai dasar pengambilan keputusan.

Akan tetapi, sebuah sistem mungkin saja dimulai dari pencatatan secara manual menggunakan buku. Kekeliruan dalam mencatat sangat mungkin terjadi.

Kemudian, dilakukanlah komputerisasi supaya dapat bekerja lebih cepat dan menghasilkan data yang valid, dengan kekeliruan yang lebih sedikit. User dapat menggunakan software yang sudah terpasang, mudah ditemukan, terjangkau, atau sesuai rekomendasi.

Berikutnya, kebutuhan bertambah. Bisa berupa kolom baru, bisa berupa aturan validasi baru, dan bertambahnya ekspektasi user.

Setelah beberapa waktu berjalan, ada yang perlu dirombak, untuk menyesuaikan dengan kebijakan perusahaan. Kolom baru lagi, aturan validasi baru lagi, dan seterusnya.

Migrasi ke software tertentu ataupun pengembangan secara khusus kemudian dilakukan, untuk mengakomodir perkembangan tersebut. Masalahnya, software tersebut memiliki struktur data tersendiri, yang mungkin tidak sepenuhnya kompatibel dengan struktur data dari software sebelumnya. Otomatis, data pun perlu dimigrasikan. Sementara, operasional terus berjalan.

Seiring waktu, rupanya software tersebut, kendati lebih canggih, lebih modern, lebih compliant, mungkin tidak dapat sepenuhnya menggantikan semua fitur di software sebelumnya. Atau, setidaknya begitulah persepsi atau alur kerja user.

Apa yang kemudian dilakukan? Apabila software-software tersebut dapat import/export data dari/ke format yang lebih umum, sebagian user mungkin mengolah data secara terpisah di software lain. Lalu, kompleksitas pun bertambah.

Berikutnya? Migrasi ke software lain lagi? Mengolah data secara terpisah dengan software tambahan? Menggunakan beberapa software sekaligus? Adanya kebijakan tertentu dari perusahaan? Anda tinggal sebut saja.

Lalu, bayangkanlah ini terjadi pada perusahaan atau entitas dimana ukurannya berkali-kali lipat.

Yang pasti, data semakin banyak dan kompleks. Masalahnya, seberapa dapat dipercaya data tersebut?

Setiap sistem menghasilkan data, tapi tidak otomatis menghasilkan data yang berkualitas.

Banyak tantangan justru muncul setelah data terkumpul.

Sumber masalah yang umum

Dalam praktiknya, data hampir tidak pernah 100% bersih, kecuali sistemnya sangat sederhana, tidak berubah, digunakan oleh user yang sangat disiplin, dan software yang digunakan sangat mumpuni dan kompatibel.

Umumnya, ada saja potensi ketidaksempurnaan sistem dan masalah pada data. Berikut adalah contoh-contoh sumber masalah yang umum.

Input manual

Paling mudahnya, ini berupa salah ketik. Sebagaimana dibahas pada bab sebelumnya, kopi, yang hanya empat huruf, bisa saja salah diketikkan.

Lalu, bisa saja, data yang harusnya diisikan, malah kosong atau terlewat.

Di bab sebelumnya, kita juga melihat kemungkinan format yang tidak konsisten.

Proses data secara manual

Bagaimana kalau keliru copy-paste ketika memroses secara manual menggunakan software?

Duplikasi data bisa terjadi karena copy-paste tersebut.

Rumus bisa saja terhapus atau berubah.

Belum lagi kalau masing-masing user kemudian bisa memiliki versi file masing-masing.

Migrasi sistem/software

Sebagaimana dibahas sebelumnya, antara software satu dan lainnya, struktur data yang digunakan sangat mungkin berbeda. Menggunakan struktur data terstandarisasi? Bagaimana kalau software tersebut memroses dengan langkah tambahan atau menginterpretasikan secara tidak standar?

Lalu, ketika migrasi data pada akhirnya dilakukan, sebagian data bisa saja hilang atau keliru dipetakan.

Kita juga tidak dapat mengabaikan kemungkinan bahwa sebagian data tercatat di software utama, sebagian dengan software tambahan, dan sebagian dicatat manual.

Perubahan kebijakan atau proses bisnis

Sebelumnya, kita membahas bahwa kita perlu menyesuaikan dengan kebijakan perusahaan. Tapi dinamika di lapangan bisa saja cukup menantang. Cara

pencatatan mungkin berbeda. Beberapa departemen atau cabang mungkin menerapkan aturan tertentu.

Apabila kita berhubungan langsung dengan pelanggan, bagaimana kalau terdapat perubahan kebijakan dalam mencatat pelanggan? Sebagian dengan email, sebagian dengan nomor telepon, sebagian tercatat lebih dari sekali. Nama yang sama juga dapat ditulis dengan berbagai cara.

Masalah teknis

Jenis masalah yang ini jangan dilupakan. Yang pertama adalah sistem mengalami downtime. Bisa saja, bukan?

Oleh karena itu, sebisa mungkin arsitektur sistem dibuat tangguh (resilient) terhadap serangan, dapat membagi beban kerja, dan dirancang untuk highly available. Skalabilitas juga dipikirkan, misal dapat menambah jumlah server (horizontal).

Masalah berikut: ketika sistem satu dan lainnya perlu melakukan sinkronisasi, mungkin saja terdapat kendala, yang memungkinkan gagalnya sinkronisasi.

Masalah teknis seperti sensor yang rusak atau tidak terkalibrasi juga menambah kemungkinan terjadinya data yang tidak bersih.

Pembersihan data

Kita telah melihat bahwa ketika sistem berjalan menghasilkan data, data tersebut mengandung noise. Sebelum layak untuk dianalisis, noise ini perlu dibersihkan.

Dalam berbagai proyek data, umum ditemukan bahwa porsi besar pekerjaan dilakukan pada memahami dan membersihkan data yang berasal dari berbagai sistem dan proses operasional.

Di bab sebelumnya, kita telah melihat berbagai contoh noise pada data. Setelah bab memuat data berikut, kita akan melihat contoh-contoh dasar bagaimana membersihkan data. Setelah itu, barulah dapat dilakukan analisis.

Memuat Data

Mari kita kembali pada data penjualan yang tidak bersih di awal buku ini. Anggap saja, data tersebut diketik menggunakan sebuah program spreadsheet, yang kemudian disimpan sebagai format file aslinya. Dengan demikian, apabila kita menggunakan nama “data”, nama filenya bisa berupa: data.xlsx, data.ods, atau data.numbers, tergantung program yang digunakan.

Di bab ini, kita akan menggunakan format file CSV, yang merupakan singkatan dari Comma-Separated Values, dengan ekstensi nama file default adalah .csv. Format ini adalah berupa file teks (sehingga dapat dibuka dengan program editor teks), dimana setiap baris (record) disimpan pada satu baris tersendiri, dan setiap kolom (field) pada baris (record) tersebut dipisahkan oleh karakter pemisah tertentu (standarnya adalah karakter koma).

Umumnya, program spreadsheet dapat menyimpan / export ke format CSV tersebut. Walau, program spreadsheet yang digunakan mungkin akan menggunakan pemisah field berupa karakter titik koma, apabila pengaturan region (locale) sistem menggunakan karakter koma sebagai pemisah desimal.

Untuk mengikuti pembahasan dalam buku ini, Anda tentu saja dapat mengetikkan sama persis dengan data penjualan yang tidak bersih tersebut, menggunakan

program spreadsheet, dan menyimpan / export sebagai format CSV dengan nama file data.csv.
Atau, Anda dapat mengetikkan/mengopikan isi file CSV berikut, menggunakan editor teks, dan menyimpannya sebagai data.csv.

```
waktu,produk,jumlah,harga
2026-06-08 08:40:50,Kopi,2,5000
2026-06-08 08:40:50,bakwan,1,6000
2026-06-08 09:05:08,bakwan,4,Rp 6000
2026-06-08 10:30:20,kopi,,5000
2026-06-08 10:50:35,Sinkong     Keju,1,40000
08/06/2026 12:04:29,bakwan,1,6000
2026-06-08 12:08:20,kpi,1,5000
2026-06-08 14:50:40,kopi,4,5000
2026-06-08 16:08:55,Bak wan,6,6000
2026-06-08 16:20:58,bakwan,3,
```

Catatan: apabila data.csv Anda menggunakan titik koma (misal ketika dibuka dengan editor teks), biarkanlah demikian.

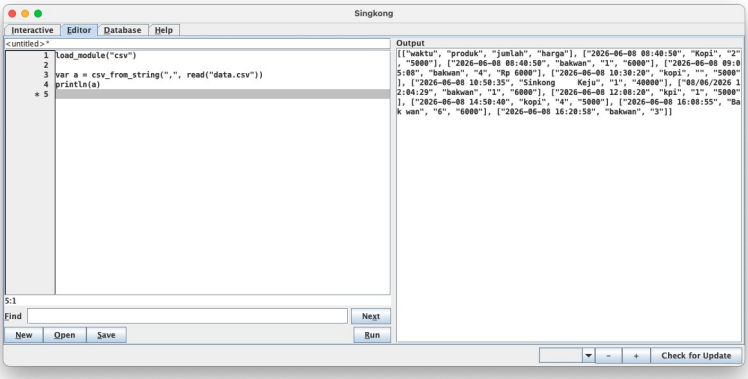
Sebelum dapat dibersihkan, data tersebut perlu dimuat.

Di Singkong, kita cukup menggunakan fungsi read untuk membaca isi file teks (sebagai STRING), dan fungsi csv_from_string (dari modul csv) untuk mengubah STRING (dalam format CSV) ke ARRAY.

Downloadlah Singkong.jar dan tempatkanlah pada direktori yang sama yang berisikan file data.csv. Jalankanlah Singkong.jar dari direktori tersebut. Kemudian, di tab Editor, ketikkanlah atau paste-lah kode berikut. Ubahlah argumen pertama csv_from_string menjadi “;” apabila data.csv menggunakan pemisah field tersebut.

```
load_module("csv")

var a = csv_from_string(",", read("data.csv"))
println(a)
```

Kemudian, dengan tombol Save, simpanlah sebagai 1_load_data_csv.singkong. Selanjutnya, tekanlah tombol Run. Berikut adalah contoh layarnya.



Apabila lebih menyukai command line, jalankanlah terminal emulator, aktiflah di direktori yang berisi Singkong.jar, data.csv, dan 1_load_data_csv.singkong, dan jalankanlah perintah berikut:

```
java -DSINGKONG=0 -jar Singkong.jar 1_load_data_csv.singkong
```

Berikut adalah contoh outputnya. Properti -DSINGKONG=0 akan meminta Singkong untuk berjalan di mode teks walaupun GUI tersedia.

```
[["waktu", "produk", "jumlah", "harga"], ["2026-06-08
08:40:50", "Kopi", "2", "5000"], ["2026-06-08
08:40:50", "bakwan", "1", "6000"], ["2026-06-08
09:05:08", "bakwan", "4", "Rp 6000"], ["2026-06-08
```

```
10:30:20", "kopi", "", "5000"], ["2026-06-08 10:50:35",
"Sinkong     Keju", "1", "40000"], ["08/06/2026
12:04:29", "bakwan", "1", "6000"], ["2026-06-08
12:08:20", "kpi", "1", "5000"], ["2026-06-08 14:50:40",
"kopi", "4", "5000"], ["2026-06-08 16:08:55", "Bak
wan", "6", "6000"], ["2026-06-08 16:20:58", "bakwan",
"3"]]
```

Apa yang berhasil kita baca adalah sebuah ARRAY dari ARRAY, yang mewakili ARRAY dari setiap baris dalam data kita. Misal:

```
["waktu", "produk", "jumlah", "harga"]
```

Atau:

```
["2026-06-08 08:40:50", "Kopi", "2", "5000"]
```

Normalnya, setiap baris terdiri dari empat kolom. Tapi, rupanya baris terakhir tidak demikian:

```
["2026-06-08 16:20:58", "bakwan", "3"]
```

Tapi, yang pasti, data telah berhasil kita muat, dan siap untuk melewati tahapan pembersihan.

Tunggu dulu. Sebelum melangkah lebih jauh, Anda mungkin bertanya, bagaimana kalau tadinya kita gagal membaca file dengan fungsi read? Misal file tidak ditemukan atau kita tidak memiliki hak akses? Maka fungsi tersebut akan mengembalikan null. Perhatikanlah apabila testing.txt memang tidak ada di direktori aktif:

```
> var s = read("testing.txt")
> type(s)
"NULL"
> println(s)
null
```

Oleh karena itu, ceklah selalu dengan if, apabila suatu fungsi mungkin mengembalikan null. Karena, apabila tidak, dalam contoh read ini, maka csv_from_string juga akan mengembalikan null (karena gagal baca sesuai aturan CSV, untuk null yang dilewatkan). Berikutnya, karena mengembalikan null, kita tidak dapat lanjutkan untuk diproses. Dan seterusnya. Pada prinsip fail fast, ketika file gagal dibaca (padahal kita butuh), kita dapat tampilkan pesan kesalahan dan hentikan program.

Malas cek? Gunakanlah ifnull, apabila sebuah nilai mungkin berupa null, dan Anda ingin mendapatkan nilai default tertentu. Contoh:

```
> var s = null
> var a = ifnull(s, "Testing")
> a
"Testing"
> ifnull(fn(){return null}(), "Testing")
"Testing"
> ifnull(fn(){return "OK"}(), "Testing")
"OK"
```

Apabila file CSV yang akan dibaca selalu menggunakan pemisah field koma, maka fungsi csv_from_string_default dapat digunakan.

```
> load_module("csv")
> var s = read("data.csv")
> csv_from_string(",", s) == csv_from_string_default(s)
true
```

Sekarang, kita siap untuk membersihkan data.

Membersihkan Data

Keseluruhan dari program membersihkan data kita akan disimpan pada file 2_clean_data.singkong. Terdapat total 79 baris, dan kita akan bahas dari awal sampai akhir, secara terurut, sedikit demi sedikit. Paste-lah ke dalam editor Singkong.

Kita mulai dari baris 1 sampai 6. Modul csv tetap di load untuk membaca dari data.csv. Lalu, kita siapkan masing-masing variabel untuk menampung nama-nama produk yang valid (dalam huruf kecil semuanya). Dan, kita akan memiliki sebuah ARRAY dengan nama PRODUCTS, yang menampung semua produk yang kita miliki.

```
load_module("csv")

var P_BAKWAN = "bakwan"
var P_KOPI = "kopi"
var P_SING_K = "singkong keju"
var PRODUCTS = [P_BAKWAN, P_KOPI, P_SING_K]
```

Baris 7 sampai 9 berikut hanyalah berupa capturing group regex untuk mendapatkan angka saja dari teks. Ya, dugaan Anda benar. Ini untuk “Rp 6000”, dimana kita hanya butuh teks “6000” nya. Belum kita gunakan.

```
var NUM = "(\d+)"
```

Baris 10 sampai 14 berikut akan kita bahas mendetil.

```
var typo = {"kpi": P_KOPI, "sinkongkeju": P_SING_K}
each(PRODUCTS, fn(e, i) {
    typo + {replace(e, " ", ""): e}
})
```

Kita tahu bahwa terdapat sejumlah salah ketik pada nama produk. Variabel typo adalah sebuah HASH, dimana kemungkinan kesalahan kita jadikan sebagai key, dalam huruf kecil, tanpa adanya spasi sama sekali. Valuenya tentunya adalah nama produk yang valid.

Anda akan bertanya: di mana bakwan? Dalam data kita, ini tidak perlu, kendati ada “Bak wan”, karena ketika memroses, selain kita akan buang semua spasi awal dan akhir, kita bahkan akan menghapus semua spasi pada nama produk.

Luar biasa bukan? Selain kesannya rumit, bagaimana kalau ada baris yang benar mengandung “singkong keju”? Padahal ini ditulis tepat sesuai nama produk yang valid. Karena kita hapus semua spasi, malah menjadi “singkongkeju”. Produk yang diketik benar, malah menjadi tidak benar, dan lebih parahnya, tidak ada key nya dalam HASH typo.

Begitulah penulis program ini. Nama bahasanya saja “Singkong”, yang lebih panjang dibandingkan sejumlah nama bahasa lain. Apalagi ekstensi nama filenya (walaupun, Anda bebas menggunakan .txt atau .sk, tidak harus .singkong).

Sebagai konsekuensi dari menghapus spasi ini, kita tambahkan HASH typo secara dinamis dengan nama produk yang valid, dimana key adalah versi tanpa spasi, dan valuenya adalah versi lengkap nama produk. Di Singkong, kita bisa tambahkan HASH dengan HASH lain (dan urutannya akan diingat).

Perhatikanlah contoh berikut:

```
> var h = {1: "satu", 2: "dua", 3: "tiga"}
> h + {4: "empat"}
{1: "satu", 2: "dua", 3: "tiga", 4: "empat"}
> h + {5: "lima"}
{1: "satu", 2: "dua", 3: "tiga", 4: "empat", 5: "lima"}
> h + {6: "enam", 7: "tujuh"}
{1: "satu", 2: "dua", 3: "tiga", 4: "empat", 5: "lima",
6: "enam", 7: "tujuh"}
> h
{1: "satu", 2: "dua", 3: "tiga", 4: "empat", 5: "lima",
6: "enam", 7: "tujuh"}
```

Penulis mengopikan sementara ke file lain, dan menambahkan println dan exit (Anda tidak perlu melakukannya), sehingga isi typo setelah each dijalankan akan menjadi:

```
{"kpi": "kopi", "sinkongkeju": "singkong keju",
"bakwan": "bakwan", "kopi": "kopi", "singkongkeju":
"singkong keju"}
```

Fungsi replace yang kita gunakan akan mengganti substring dalam sebuah STRING:

```
> replace("singkong keju", " ", "")
"singkongkeju"
```

Di kode program Singkong, each akan cukup sering ditemukan untuk perulangan. Perhatikanlah contoh berikut (dimana e akan berupa elemennya, dan i akan

berupa indeksnya, untuk setiap kali perulangan dilakukan):

```
var a = ["bahasa", "Singkong"]
each(a, fn(e, i) {
    println("indeks " + i + " : " + e)
})
```

Keluarannya adalah:

```
indeks 0 : bahasa
indeks 1 : Singkong
```

Dengan demikian, `{replace(e, " ", ""): e}` akan berupa sebuah HASH, dimana key nya adalah versi tanpa spasi, dan valuenya adalah versi lengkap produk, sebagaimana disebutkan sebelumnya.

Antisipasi kekeliruan nama produk memang tidak ideal atau mencakup semua kemungkinan, tapi untuk data kita, sudah cukup.

Baris 15 dan 16 berikut telah dibahas di bab sebelumnya:

```
var a = csv_from_string(",", read("data.csv"))
```

Baris 17 sampai 35 berikut adalah bagian paling sulit dari program kita di bab ini. Penulis bukannya sengaja mempersulit. Hanya saja, karena kolom waktu adalah kolom pertama, maka penulis membuat fungsi membersihkan waktu ini duluan.

```
var clean_datetime = fn(s) {
    var s = trim(ifnull(s, ""))
    if (empty(s)) {
        return null
    }
```

```
    if (in(s, "/")) {
        var x = split(s, "/")
        if (len(x) == 3) {
            var t = split(x[2], " ")
            if (len(t) == 2) {
                var tt = t[0] + "-" + x[1] + "-"
                    + x[0] + " " +  t[1]
                return datetime(tt, @, true)
            }
        }
    }
    return datetime(s, @, true)
}
```

Fungsi ini menerima satu parameter, yaitu STRING penanda waktu, misal: “2026-06-08 08:40:50”, “08/06/2026 12:04:29”, atau kemungkinan format lainnya. Dengan trim, kita buang whitespace (spasi, tab, ...) di depan dan di belakang STRING tersebut. Penggunaan ifnull akan memastikan argumen yang dilewatkan, apabila null, akan menjadi `""`. Pada akhirnya, apabila setelah di trim hasilnya adalah `""` (empty akan memeriksa ini, selain untuk HASH dan ARRAY), maka fungsi akan mengembalikan null. Kesepakatan kita adalah, apabila null, artinya ada yang tidak benar sesuai aturan kita.

```
    var s = trim(ifnull(s, ""))
    if (empty(s)) {
        return null
    }
```

Setelah itu, kita cek, apakah mengandung STRING “/”. Kita asumsikan, mungkin terdapat cara penulisan dengan format: tanggal (2 digit)/bulan (2 digit)/tahun (4 digit) diikuti spasi, lalu jam (24 jam, 2 digit):menit (2 digit):detik (2 digit). Seperti pada “08/06/2026 12:04:29”.

```
if (in(s, "/")) {
}
```

Apabila ya, kita akan memecah STRING tersebut berdasarkan STRING “/”.

```
var x = split(s, "/")
```

Contoh hasil dari pemanggilan tersebut adalah ARRAY berikut (kode penulis ketikkan pada interactive evaluator untuk ilustrasi, Anda tidak perlu mengetikkannya):

```
> var s = "08/06/2026 12:04:29"
> var x = split(s, "/")
> x
["08", "06", "2026 12:04:29"]
```

Kita mengharapkan sebuah ARRAY dengan tiga elemen. Oleh karena itu, kita cek panjangnya dengan len:

```
if (len(x) == 3) {
}
```

Apabila sesuai yang kita harapkan, maka, kita akan proses elemen ke-2 (karena, indeks ARRAY dimulai dari 0), yang merupakan sebuah STRING dengan contoh nilai berikut: "2026 12:04:29". Kita split lagi (dengan pemisah adalah satu spasi) untuk mendapatkan tahun (“2026”) dan jam:menit:detik (“12:04:29”), dan pastikan hasilnya adalah ARRAY dengan dua elemen:

```
var t = split(x[2], " ")
if (len(t) == 2) {
}
```

Kita tinggal susun ulang semua bagian dari penanda waktu tersebut menjadi sama seperti baris data lainnya, yaitu tahun (4 digit)-bulan (2 digit)-tanggal (2 digit) diikuti

spasi, lalu jam (24 jam, 2 digit):menit (2 digit):detik (2 digit).

```
var tt = t[0] + "-" + x[1] + "-"
    + x[0] + " " +  t[1]
```

Pada contoh “08/06/2026 12:04:29”, ini akan menjadi “2026-06-08 12:04:29”.

Apapun STRING yang kita dapatkan, kita akan lewatkan ke fungsi datetime, yang akan melakukan konversi ke tipe DATE di Singkong (tipe ini dimaksudkan sebagai penanda waktu lengkap, mulai dari tahun sampai detik, dengan nilai 0 diasumsikan apabila bagian waktunya tidak lengkap). Memang kita berikan nilai default berupa tanggal/jam sekarang (diwakili oleh @ di Singkong) apabila konversi gagal (argumen ke-2). Tapi nilai DATE apapun boleh (misal @2026), karena, untuk argumen ke-3, kita berikan true. Ini diartikan sebagai: apabila konversi ke DATE gagal, fungsi akan mengembalikan null (bukan nilai default). Contoh konversi yang gagal:

```
> datetime("test")
2026-07-01 14:41:33
> datetime("test", @2026)
2026-01-01 00:00:00
> datetime("test", @202607)
2026-07-01 00:00:00
> datetime("test", @, false)
2026-07-01 14:42:08
> datetime("test", @, true)
> type(datetime("test", @, true))
"NULL"
```

Contoh konversi yang berhasil:

```
> var s = string(@)
> s
"2026-07-02 17:48:53"
```

```
> var d = datetime("2026-07-02 17:48:53")
> d
2026-07-02 17:48:53
> type(d)
"DATE"
```

Kembali ke fungsi kita. Apapun hasilnya, kita return.

```
return datetime(tt, @, true)
```

Bagaimana kalau terdapat cara tulis penanda waktu lainnya? Untuk kali ini, kita cukup lewatkan ke fungsi datetime seperti sebelumnya. Memang akan berpotensi mengembalikan null. Tidak ideal, tapi cukup untuk data kita.

```
    return datetime(s, @, true)
```

Mari kita lanjut ke baris 36 sampai 47. Pastinya, setelah mempersiapkan cara menangani salah ketik dan contoh fungsi sebelumnya, ini akan terasa lebih mudah.

```
var clean_product = fn(s) {
    var s = replace(lower(trim(ifnull(s, ""))), " ", "")
    if (in(PRODUCTS, s)) {
        return s
    }
    var t = typo[s]
    if (t != null) {
        return t
    }
    return null
}
```

Baris pertama dari fungsi ini mungkin terlihat merepotkan.

```
    var s = replace(lower(trim(ifnull(s, ""))), " ", "")
```

Yang kita lakukan adalah:

- Menggunakan ifnull dan trim seperti fungsi sebelumnya. Apabila fungsi menerima “Bak wan”, maka sampai di sini, akan tetap “Bak wan”.
- Menjadikannya huruf kecil dengan fungsi lower. Sehingga, “Bak wan” akan menjadi “bak wan”.
- Fungsi replace, yang telah dicontohkan sebelumnya, akan menghapus spasi, sehingga yang terakhir “bak wan” akan menjadi “bakwan”.

Berikutnya, kita tinggal cek:

```
    if (in(PRODUCTS, s)) {
        return s
    }
```

Apakah kita mendapatkan STRING yang nilainya ada dalam ARRAY PRODUCTS? Mari kita lihat untuk contoh “bakwan”:

```
var P_BAKWAN = "bakwan"
var PRODUCTS = [P_BAKWAN, P_KOPI, P_SING_K]
```

Ya, sebagaimana Anda lihat, fungsi in akan mengembalikan true. Dan, nilai “bakwan” ini akan direturn. Sudah bersih dan valid sebagai nama produk.

Untuk sekedar mencontohkan penggunaan fungsi in:

```
> in(["bakwan", true, null, 12345, [], {}], "bakwan")
true
> in(["bakwan"], 12345)
false
> in("singkongkeju", "singkong")
true
> in(keys(h), 5)
true
> in(keys(h), 6)
false
> in(values(h), 5)
false
```

```
> in(values(h), 6)
true
```

Bagaimana dengan nilai lain yang lolos? Misal:"sinkongkeju". Di baris 10, kita sudah antisipasi ini. Sampai setelah baris ke-14, isi variabel typo kita, penulis tuliskan ulang, adalah demikian:

```
{"kpi": "kopi", "sinkongkeju": "singkong keju",
"bakwan": "bakwan", "kopi": "kopi", "singkongkeju":
"singkong keju"}
```

Kode berikut akan mencoba mendapatkan value dari HASH typo, berdasarkan nilai produk yang dilewatkan:

```
    var t = typo[s]
```

Lihatlah contoh kode berikut:

```
> var typo = {"kpi": "kopi", "sinkongkeju": "singkong
keju", "bakwan": "bakwan", "kopi": "kopi",
"singkongkeju": "singkong keju"}
> var s = "sinkongkeju"
> var t = typo[s]
> t
"singkong keju"
> typo["testing"]
> println(typo["testing"])
null
```

Apabila tidak ada key yang valid, null akan dikembalikan. Pada "sinkongkeju", variabel t akan bernilai true (dan bukan null). Oleh karena itu, juga sudah bersih dan valid sebagai nama produk. Kita return.

```
    if (t != null) {
        return t
    }
```

Karena semua kemungkinan nama produk valid sudah kita tentukan, maka di luar itu, kita cukup return null.

Sebagaimana kesepakatan sebelumnya, ada yang tidak benar sesuai aturan kita:

```
    return null
```

Tentu saja, seperti ketika membersihkan kolom waktu sebelumnya, fungsi kita masih tidak ideal, karena semua kemungkinan salah ketik tidak dapat kita akomodir. Tapi, sekali lagi, untuk data kita, ini sudah cukup.

Sampailah kita pada kolom jumlah, untuk kuantitas produk yang kita jual. Baris 48 sampai 56 berikut tidaklah panjang, dan juga tidak sulit:

```
var clean_qty = fn(s) {
    var s = replace(lower(trim(ifnull(s, ""))), " ", "")
    var a = extract(s, NUM)
    if (empty(a)) {
        var a = ["1"]
    }
    return number(a[0], 0, true)
}
```

Baris replace dan lower sama saja dengan fungsi sebelumnya.

Ingatkah Anda dengan baris 8 ini?

```
var NUM = "(\d+)"
```

Kita akan gunakan dalam pemanggilan fungsi extract berikut:

```
    var a = extract(s, NUM)
```

Artinya, ambil angka saja (\d+) dalam STRING yang dilewatkan. Perhatikanlah contoh berikut:

```
> var NUM = "(\d+)"
> var s = "Rp 6000"
> var a = extract(s, NUM)
> a
["6000"]
```

Bagaimana kalau STRING yang dilewatkan adalah: "Pembeli sangat menyukai singkong sehingga kita menjual 80 porsi @40000"? Pastilah kasirnya sangat bersemangat ketika mengetikkan teks tersebut (harusnya, kan cukup mengetikkan 80).

```
> var s = "Pembeli sangat menyukai singkong sehingga
kita menjual 80 porsi @40000"
> extract(s, NUM)
["80", "40000"]
```

Lihatlah, kita tetap mendapatkan angkanya.

Fungsi extract akan mengekstrak STRING berdasarkan pola regular expression. Apabila Anda tertarik lebih lanjut, fungsi matches dan extract_group juga tersedia di Singkong. Terdapat beberapa contoh penggunaan pada dokumentasi bahasa Singkong.

Mari kita lanjutkan ke contoh lain. Bagaimana kalau kasirnya terburu-buru ingin mengabari yang memasak, sehingga lupa mengisikan jumlah?

```
> var s = ""
> extract(s, NUM)
[]
```

Kita akan mendapatkan ARRAY kosong. Tapi, apa yang kita lakukan di baris-baris berikut?

```
    if (empty(a)) {
        var a = ["1"]
    }
```

Kalau ARRAY-nya kosong (cek dengan fungsi empty), kita buat ARRAY-nya berisikan sebuah elemen, berupa “1”. Kita melakukan imputasi data. Sebagaimana dibahas di awal buku ini, kita menebak dengan jumlah minimal 1.

Di fungsi ini, kita mengasumsikan hanya elemen pertama (indeks 0) dalam ARRAY yang akan kita proses. Jadi, ["6000"] dan ["1"] sesuai yang kita harapkan, tapi tidak dengan ["80", "40000"]. Seperti sebelumnya, tidak ideal juga, tapi kemungkinan deretan teks tersebut diisikan sangatlah kecil, sesuka apapun dengan “singkong”.

Anda mungkin bertanya, kenapa “1” (STRING) dan bukannya 1 (NUMBER) saja? Karena, kita akan gunakan fungsi number, yang akan mengkonversi STRING atau DATE ke NUMBER. Sebagaimana halnya pada datetime, kita memberikan nilai default, tapi tidak akan digunakan, karena contoh berikut melewatkan true (kembalikan null apabila konversi gagal) ketika memanggil number:

```
    return number(a[0], 0, true)
```

Perhatikanlah contoh berikut:

```
> number("test")
0
> number("test", 12345)
12345
> number("test", 12345, false)
12345
> type(number("test", 12345, true))
"NULL"
> number("6000", 0, true)
```

```
6000
> type(number("6000", 0, true))
"NUMBER"
```

Kita akan mendapatkan NUMBER atau null. Kalau null, sebagaimana kesepakatan sebelumnya, ada yang tidak benar sesuai aturan kita.

Fungsi terakhir kita adalah untuk membersihkan STRING harga jual. Berikut adalah baris 57 sampai 65.

```
var clean_price = fn(s) {
    var s = replace(lower(trim(ifnull(s, ""))), " ", "")
    var a = extract(s, NUM)
    if (empty(a)) {
        return null
    }
    return number(a[0], 0, true)
}
```

Sangat mirip dengan fungsi sebelumnya, bukan? Tentu saja tujuan penulis bukanlah untuk membuat kode program kita lebih panjang dan buku ini sedikit lebih tebal.

Bedanya hanya, kalau extract mengembalikan ARRAY kosong, karena gagal mendapatkan angka, yang artinya harga jual tidak diisikan, kita segera kembalikan null. Sebagaimana dibahas di awal buku, untuk contoh ini, kita tidak menebak karena harga bersifat sensitif terhadap omzet dan laba rugi bisnis.
Program kita berjumlah 79 baris. Sisa sedikit lagi. Baris kode 66 sampai 78 berikut pada dasarnya mengulang baris demi baris data, dan untuk setiap kolomnya,

melewati proses pembersihan dengan fungsi-fungsi yang kita buat.

```
var r = []
each(a, fn(e, i) {
    if (i > 0) {
        var d = clean_datetime(e[0])
        var p = clean_product(e[1])
        var q = clean_qty(e[2])
        var s = clean_price(e[3])
        var c = [d, p, q, s]
        if (!in(c, null)) {
            r + c
        }
    }
})
```

Variabel r, sebuah ARRAY, awalnya adalah ARRAY kosong.

```
var r = []
```

Kemudian, untuk setiap elemen dalam ARRAY a (ARRAY dari ARRAY) yang kita baca di baris 15 dengan fungsi csv_from_string, kita hanya proses untuk indeks lebih besar dari 0. Kenapa? Karena, indeks ke 0 adalah ARRAY berikut:

```
["waktu", "produk", "jumlah", "harga"]
```

Oleh karena itu, kita cek indeks yang diwakili dengan variabel i dalam hal ini:

```
    if (i > 0) {
    }
```

Empat baris berikut adalah pemanggilan empat fungsi yang kita bahas sebelumnya, untuk masing-masing kolom. Perhatikanlah bahwa variabel e sendiri adalah sebuah ARRAY, karena a adalah sebuah ARRAY dari

ARRAY. Jadi, setiap elemen dalam a adalah sebuah ARRAY. Artinya, indeks ke-0 adalah waktu, ke-1 adalah produk, ke-2 adalah jumlah, dan ke-3 adalah harga.

```
var d = clean_datetime(e[0])
var p = clean_product(e[1])
var q = clean_qty(e[2])
var s = clean_price(e[3])
```

Apapun hasil dari pemanggilan masing-masing fungsi tersebut, kita buat sebuah ARRAY baru, dengan nama variabel c:

```
var c = [d, p, q, s]
```

Untuk apa? Untuk if yang canggih berikut.

```
if (!in(c, null)) {
    r + c
}
```

Dan, menurut Anda, kenapa if tersebut canggih? Kerena, kita tinggal cek, dengan fungsi in, apakah ARRAY c yang baru dibuat, mengandung null atau tidak. Sekali lagi, sebagaimana kesepakatan sebelumnya, null artinya ada yang tidak benar sesuai aturan kita. Oleh karena itu, kita hanya tambahkan ke ARRAY r apabila tidak ada null sama sekali dalam ARRAY c.

Dalam hal ini, satu-satunya baris yang mengandung null adalah baris terakhir data kita. Andaikata if kita seperti ini:

```
if (!in(c, null)) {
    r + c
} else {
    println("ERROR: " + c)
}
```

Maka, ketika dijalankan (file dikopikan sebagai test.singkong dan diedit), outputnya adalah:

```
java -DSINGKONG=0 -jar Singkong.jar test.singkong

ERROR: [2026-06-08 16:20:58, "bakwan", 3, null]
true
```

Kenapa? Karena indeks ke-3, atau harga, setelah melewati clean_price, adalah null. Karena harga jual tidak diisikan.

Lalu, andaikata setelah blok each, kita tambahkan baris berikut:

```
println(r)
```

Maka, ketika dijalankan, outputnya adalah:

```
java -DSINGKONG=0 -jar Singkong.jar test.singkong

ERROR: [2026-06-08 16:20:58, "bakwan", 3, null]
[[2026-06-08 08:40:50, "kopi", 2, 5000], [2026-06-08 08:40:50, "bakwan", 1, 6000], [2026-06-08 09:05:08, "bakwan", 4, 6000], [2026-06-08 10:30:20, "kopi", 1, 5000], [2026-06-08 10:50:35, "singkong keju", 1, 40000], [2026-06-08 12:04:29, "bakwan", 1, 6000], [2026-06-08 12:08:20, "kopi", 1, 5000], [2026-06-08 14:50:40, "kopi", 4, 5000], [2026-06-08 16:08:55, "bakwan", 6, 6000]]
true
```

Di baris terakhir program ini, baris ke-79:

```
write("data_clean.csv", csv_to_string(",", r))
```

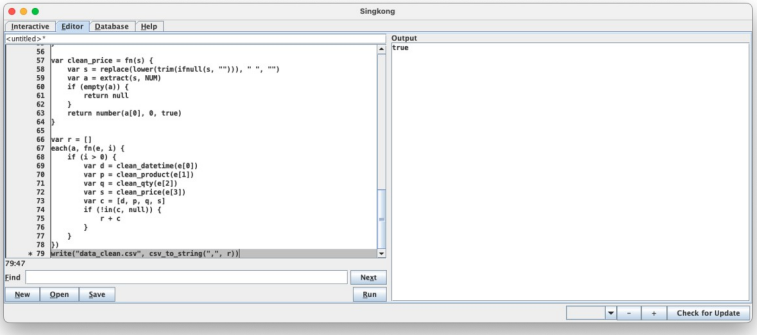
Kita melakukan kebalikan dari:

```
var a = csv_from_string(",", read("data.csv"))
```

Yaitu:

- Kita jadikan ARRAY r sebagai sebuah STRING CSV, dengan pemisah field adalah koma.
- Setelah itu, kita tinggal tulis ke file "data_clean.csv" dengan fungsi write. Apabila berhasil, fungsi akan mengembalikan true (Apakah Anda mencermati “true” pada di contoh menjalankan test.singkong di halaman sebelumnya? Supaya tidak muncul, silahkan assign pemanggilan write ke variabel).

Program kita pun selesai. Simpanlah filenya dengan menekan tombol Save pada editor Singkong. Setelah itu, kliklah tombol Run. Di panel output, true akan ditampilkan apabila data_clean.csv berhasil ditulis.



Perhatikanlah isi file data_clean.csv:

```
2026-06-08 08:40:50,kopi,2,5000
2026-06-08 08:40:50,bakwan,1,6000
2026-06-08 09:05:08,bakwan,4,6000
2026-06-08 10:30:20,kopi,1,5000
2026-06-08 10:50:35,singkong keju,1,40000
2026-06-08 12:04:29,bakwan,1,6000
2026-06-08 12:08:20,kopi,1,5000
2026-06-08 14:50:40,kopi,4,5000
2026-06-08 16:08:55,bakwan,6,6000
```

Bisa kita lihat bahwa isi file tersebut:

- Tanpa header waktu, produk, jumlah, dan harga.
- Penanda waktu menggunakan format yang kita tentukan, dan konsisten.
- Nama produk sesuai dengan ARRAY PRODUCTS.
- Untuk baris ke-3, dimana bakwan terjual 4, dengan harga satuan 6000, kini hanya tertulis 6000 (bukan Rp 6000).
- Pada baris ke-4, dimana kopi terjual dengan harga satuan 5000, jumlahnya tidak lagi kosong, tapi telah kita isikan 1.
- Baris terakhir pada data.csv (2026-06-08 16:20:58,bakwan,3,) tidak ikut disertakan karena harga satuan tidak diketahui.
- Dari 10 baris data (di luar header), kini tersisa 9.

Pada dasarnya, data kita kini telah “bersih”, untuk kebutuhan kita.

Hanya saja, catatan berikut mungkin berguna: berkali-kali, kita menyebut fungsi-fungsi yang kita buat untuk membersihkan waktu, produk, jumlah, dan harga sebagai “tidak ideal”. Alasannya, sebagaimana kita ketahui, adalah tidak bisa membersihkan berbagai “masalah” lain pada data, yang tidak kita antisipasi. Entah itu penanda waktu dengan format lain, bagian waktu yang tidak lengkap, variasi salah ketik lainnya pada nama produk, rentang jumlah dan harga yang tidak wajar (misal: menjual bakwan sejumlah -5), dan lain sebagainya.

Saat ini, perlakuan kita pada data yang gagal dibersihkan (fungsinya mengembalikan null) adalah diabaikan begitu saja.

Ketika penulis melampirkan kode program ini untuk direview pada sebuah group chat di kampus, Dr. Mahfudh Ahmad dari Satu Data Indonesia memberikan masukan yang sangat berguna, bahwa pada kode:

```
if (!in(c, null)) {
    r + c
}
```

agar jangan langsung membuang yang tidak bersih, tapi misal disimpan pada file tersendiri, untuk dijadikan referensi lebih lanjut dalam membersihkan data.

Program kita memang hanya menyimpan yang berhasil dibersihkan. Untuk buku ini, kita kehilangan satu dari sepuluh baris. Tapi, di dunia nyata, data bisa didapatkan dari berbagai tahapan migrasi sistem dan hasil penggabungan. Dengan jumlah data jutaan baris misalnya, yang gagal dibersihkan bisa jauh lebih banyak.

Dengan menyimpan yang gagal dibersihkan untuk dijadikan referensi, proses membersihkan data bisa terus dievaluasi dan diperbaiki.

Sekarang, kita siap untuk melanjutkan pembahasan ke tahap analisis di bab berikut.

Analisis Data

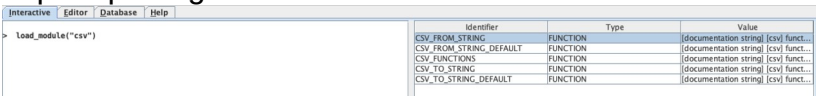
Keseluruhan dari program membersihkan data kita akan disimpan pada file 3_analyze_data.singkong. Terdapat total 136 baris, dan sama seperti sebelumnya, kita akan bahas dari awal sampai akhir, secara terurut, sedikit demi sedikit. Paste-lah ke dalam editor Singkong.

Jumlah barisnya lebih banyak, karena kita juga membuat dashboard GUI sederhana, langsung pada file yang sama. Dengan demikian, fungsi-fungsi seperti load, load_file_or_resource, dan load_resource tidak diperlukan.

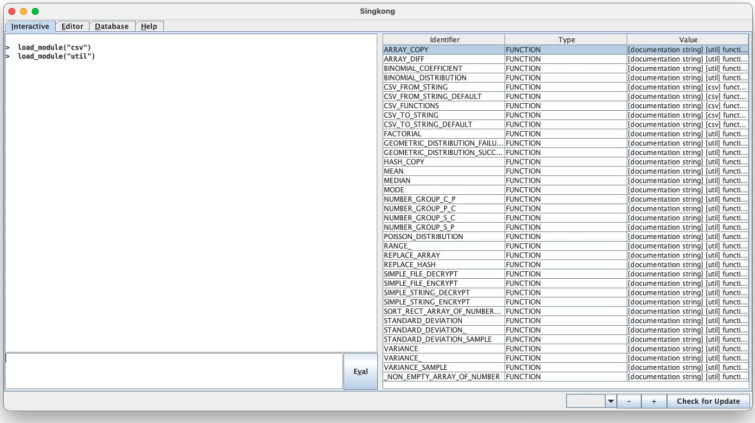
Kita mulai dari baris 1 sampai 4. Modul csv tetap di load untuk membaca dari data_clean.csv. Kali ini, kita juga load modul util untuk fungsi-fungsi tambahan seperti mean, median, mode, range_, dan standard_deviation.

```
load_module("csv")
load_module("util")

var a = csv_from_string(",", read("data_clean.csv"))
```

Sebelum melanjutkan, penulis ingin memperkenalkan fungsi variables di Singkong, untuk mendapatkan keseluruhan variabel global. Cobalah menjalankan Singkong, dan load lah modul csv. Apabila sebelumnya belum ada variabel, maka daftar identifier akan tampil seperti pada gambar berikut.



Ketika modul util di load, daftar identifier ini akan bertambah, seperti gambar berikut.



Apabila Anda ingin mengetahui saat ini ada variabel global apa saja (termasuk dari hasil load modul dan variabel yang dibuat sendiri), fungsi variables dapat digunakan. Fungsi mengembalikan HASH, dengan key adalah nama variabel dan value adalah ARRAY [tipe, value]. Perhatikanlah contoh berikut:

```
> keys(variables())
["ARRAY_COPY", "ARRAY_DIFF", "BINOMIAL_COEFFICIENT",
"BINOMIAL_DISTRIBUTION", "CSV_FROM_STRING",
"CSV_FROM_STRING_DEFAULT", "CSV_FUNCTIONS",
"CSV_TO_STRING", "CSV_TO_STRING_DEFAULT", "FACTORIAL",
"GEOMETRIC_DISTRIBUTION_FAILURE",
"GEOMETRIC_DISTRIBUTION_SUCCESS", "HASH_COPY", "MEAN",
"MEDIAN", "MODE", "NUMBER_GROUP_C_P",
"NUMBER_GROUP_P_C", "NUMBER_GROUP_S_C",
"NUMBER_GROUP_S_P", "POISSON_DISTRIBUTION", "RANGE_",
"REPLACE_ARRAY", "REPLACE_HASH", "SIMPLE_FILE_DECRYPT",
"SIMPLE_FILE_ENCRYPT", "SIMPLE_STRING_DECRYPT",
"SIMPLE_STRING_ENCRYPT",
```

```
"SORT_RECT_ARRAY_OF_NUMBER_BY_INDEX",
"STANDARD_DEVIATION", "STANDARD_DEVIATION_",
"STANDARD_DEVIATION_SAMPLE", "VARIANCE", "VARIANCE_",
"VARIANCE_SAMPLE", "_NON_EMPTY_ARRAY_OF_NUMBER"]
```

Di Singkong, nama variabel dan fungsi tidak membedakan huruf besar dan kecil. Jadi, csv_to_string sama saja dengan CSV_TO_STRING, atau kombinasi huruf besar/kecil lainnya.

Mari kita lanjutkan programnya. Baris 5 sampai 9 adalah ARRAY-ARRAY yang akan kita gunakan nanti. Saat ini, berupa ARRAY kosong semuanya.

```
var r_a = []

var a_qty = []
var a_price = []
var a_total = []
```

Di sisi atas dashboard kita (40 persen dari tinggi dan 100 persen dari lebar frame aplikasi), sebuah tabel akan ditampilkan. Pada dasarnya, ini adalah isi dari data_clean.csv, hanya saja kita tampilkan lebih rapi. Misal, rata kanan untuk bilangan (jumlah dan harga, dengan pemisah ribuan), rata kiri untuk waktu dan nama produk. ARRAY r_a nantinya adalah isi dari tabel tersebut.

Sementara, a_qty, a_price, dan a_total adalah ARRAY-ARRAY yang akan berisikan setiap jumlah, harga, dan jumlah x harga. Kita memiliki sembilan baris data, dan oleh karena itu, masing-masing ARRAY akan berisi sembilan elemen.

Di baris 10 dan 11, kita membuat dua HASH:

```
var sold = {}
var sold_total = {}
```

Keduanya berguna untuk:

- sold: key adalah nama produk, value adalah total jumlah terjual. Setiap kali perulangan dilakukan, valuenya akan di-update.
- sold_total: sama seperti sold, tapi valuenya berupa total (jumlah x harga).

Kalau dilihat, kita memiliki tiga produk, sehingga akan terdapat tiga key dan value, untuk masing-masing sold dan sold_total.

Berikut adalah baris 12 dan 13:

```
var colors = ["red", "green", "blue", "pink", "yellow"]
```

ARRAY colors akan berisi kemungkinan warna apa saja untuk setiap bar dalam barchart kita. Benar, dashboard kita akan menyediakan dua barchart di pojok kanan bawah (60 persen dari tinggi dan 50 persen dari lebar frame aplikasi).

Baris 14, kita membuat sebuah variabel, ng, dan menjadikannya “alias” dari fungsi number_group_p_c dari modul util. Fungsi ini memformat NUMBER agar tampil dengan pemisah ribuan berupa titik (p untuk point) dan pemisah desimal berupa koma (c untuk comma). Karena nama fungsinya cukup panjang, dan kita perlu tuliskan berkali-kali, kita buat aliasnya.

Baris 15 sampai 41 memroses ARRAY a yang telah kita baca sebelumnya dari data_clean.csv. Tentu saja, ini adalah ARRAY dari ARRAY seperti dibahas pada bab membersihkan data sebelumnya.

```
each(a, fn(e, i) {
    var p = e[1]
    var q = number(e[2])
    var s = number(e[3])
    var t = q * s
    a_qty + q
    a_price + s
    a_total + t
    r_a + [
        e[0],
        upper(e[1], true),
        ng(q),
        ng(s),
        ng(t)
    ]
    var o = sold[p]
    if (o == null) {
        set(sold, p, 0)
    }
    set(sold, p, sold[p]+q)
    var ot = sold_total[p]
    if (ot == null) {
        set(sold_total, p, 0)
    }
    set(sold_total, p, sold_total[p]+t)
})
```

Ingatlah bahwa setiap elemen dalam ARRAY e, dalam perulangan yang mewakili ARRAY baris data, adalah STRING. Baik waktu, produk, jumlah, dan harga. Kita tidak analisis dari sisi waktu, sehingga tidak perlu kita ubah ke DATE. Tapi, untuk jumlah (variabel q, indeks

ke-2, kolom ke-3 setelah produk) dan harga (variabel s, indeks ke-3, kolom ke-4 setelah jumlah), kita perlu ubah ke NUMBER. Dengan demikian, kita bisa hitung total (variabel t, sebagai q * s). Untuk produk, kita gunakan variabel p (lebih nyaman ditulis daripada e[1]).

```
var p = e[1]
var q = number(e[2])
var s = number(e[3])
var t = q * s
```

Untuk setiap jumlah (q, sebuah NUMBER), harga (s, sebuah NUMBER), dan total (t, juga NUMBER), kita tambahkan ke masing-masing ARRAY a_qty, a_price, dan a_total yang sudah kita siapkan sebelumnya.

```
a_qty + q
a_price + s
a_total + t
```

Kini, sudah saatnya, kita buat ARRAY berisikan waktu (e[0]), produk (p, kita gunakan huruf besar di huruf pertama, dengan upper dan argumen true), jumlah (q), harga (s), dan total (t). Untuk yang berupa NUMBER, kita format dengan fungsi alias “ng”. ARRAY tersebut, mewakili setiap baris dalam tabel, kemudian kita tambahkan ke r_a.

```
r_a + [
    e[0],
    upper(e[1], true),
    ng(q),
    ng(s),
    ng(t)
]
```

Berikutnya, kita akan mengupdate jumlah terjual, untuk setiap produk:

```
    var o = sold[p]
    if (o == null) {
        set(sold, p, 0)
    }
    set(sold, p, sold[p]+q)
```

Awalnya, karena sold adalah HASH kosong {}, maka tentunya variabel o akan bernilai null. Berikut penulis ketikkan pada evaluator, dengan produk “bakwan”:

```
> var sold = {}
> var p = "bakwan"
> var o = sold[p]
> type(o)
"NULL"
```

Karena null, mari kita buah key dengan nama produk tersebut, dan berikan value 0 (nol). Jadi, sampai di sini, “bakwan” terjual nol.

```
> set(sold, p, 0)
{"bakwan": 0}
```

Di perulangan berikutnya, tentunya o tidak lagi bernilai null, melainkan nol:

```
> var o = sold[p]
> o
0
```

Anggaplah berikutnya, ada “bakwan” yang terjual 4. Kita tinggal tambahkan ke 0 yang tadi:

```
> var q = 4
> set(sold, p, sold[p]+q)
{"bakwan": 4}
```

Yang tadinya sold[p] adalah 0, kita tambahkan 4 sehingga nilainya menjadi 4. Lalu, kita set key “bakwan” dengan value berupa 4 tadi, dengan fungsi set.

Jadi, yang kita lakukan adalah “kalau belum tercatat, catatlah sebagai nol. Kalau sudah ada, tambahkan jumlahnya.”

Berikutnya, total dari harga jual, juga sama saja:

```
    var ot = sold_total[p]
    if (ot == null) {
        set(sold_total, p, 0)
    }
    set(sold_total, p, sold_total[p]+t)
```

Andaikata kita print variabel r_a setelah perulangan, maka isinya adalah sebagai berikut:

```
java -DSINGKONG=0 -jar Singkong.jar test.singkong

[["2026-06-08 08:40:50", "Kopi", "2", "5.000",
"10.000"], ["2026-06-08 08:40:50", "Bakwan", "1",
"6.000", "6.000"], ["2026-06-08 09:05:08", "Bakwan",
"4", "6.000", "24.000"], ["2026-06-08 10:30:20", "Kopi",
"1", "5.000", "5.000"], ["2026-06-08 10:50:35",
"Singkong keju", "1", "40.000", "40.000"], ["2026-06-08
12:04:29", "Bakwan", "1", "6.000", "6.000"], ["2026-06-
08 12:08:20", "Kopi", "1", "5.000", "5.000"], ["2026-06-
08 14:50:40", "Kopi", "4", "5.000", "20.000"], ["2026-
06-08 16:08:55", "Bakwan", "6", "6.000", "36.000"]]
```

Jadi, sebuah ARRAY dari ARRAY, dari isi file data_clean.csv. Hanya saja, nama produk diberikan huruf besar di awal (“Bakwan”, “Singkong keju”), dan yang tadinya bilangan (“5000” lalu menjadi 5000), diberikan pemisah ribuan (“5.000”). Data ini akan kita tampilkan ke tabel, begitu tabelnya siap.

Baris 42 sampai 57 adalah inti dari analisis datanya.

```
var q_mean = mean(a_qty)
var q_median = median(a_qty)
```

```
var q_mode = mode(a_qty)
var q_range = range_(a_qty)
var q_std = standard_deviation(a_qty)
var s_mean = mean(a_price)
var s_median = median(a_price)
var s_mode = mode(a_price)
var s_range = range_(a_price)
var s_std = standard_deviation(a_price)
var t_mean = mean(a_total)
var t_median = median(a_total)
var t_mode = mode(a_total)
var t_range = range_(a_total)
var t_std = standard_deviation(a_total)
```

Kita cukup memanggil fungsi-fungsi (mean, median, mode, range_, standard_deviation) yang disediakan oleh modul util.

Mean

Andaikata kita tampilkan variabel a_qty (ARRAY jumlah produk terjual), isinya adalah sebagai berikut:

```
[2, 1, 4, 1, 1, 1, 1, 4, 6]
```

Mean-nya (variabel q_mean) adalah 21/9 = 2,3333

Artinya bagi kita:

- Secara rata-rata, setiap kali ada pelanggan yang datang, mereka akan membeli antara 2 sampai 3 porsi (sekitar 2 porsi).
- Dari sisi stok, andaikata besok diperkirakan ada 10 transaksi, maka total yang akan terjual adalah sekitar 23 porsi.
- Tapi, mana ada orang membeli 2,3333 porsi bakwan? Mean sangatlah sensitif pada nilai yang

berbeda sekali, misal tiba-tiba ada yang membeli 6 porsi bakwan sekaligus. Hasilnya, sebagaimana kita lihat, berupa bilangan pecahan seperti ini. Kita tidak dapat sepenuhnya mengandalkan mean.

Bagaimana dengan variabel a_price, yang adalah ARRAY harga jual?

```
[5000, 6000, 6000, 5000, 40000, 6000, 5000, 5000, 6000]
```

Mean-nya (variabel s_mean) adalah 9.333,3333.

Artinya bagi kita:

- Setiap porsi yang terjual, pendapatan kotornya adalah Rp 9.333. Misal target hari ini adalah 100 porsi terjual, estimasi omsetnya adalah sekitar Rp 933.300.
- Tapi, apakah memang produk kita kisaran harganya adalah di Rp 9.000-an per porsi? Rupanya tidak. Kopi dijual seharga Rp 5.000 dan Bakwan Rp 6.000. Kenapa rata-ratanya bisa sampai Rp 9.333? Terdapat outlier, yaitu Singkong keju seharga Rp 40.000. Data ekstrem ini menarik nilai rata-rata harga jual ke atas, sehingga tidak lagi mencerminkan harga mayoritas produk.
- Apabila hanya mengandalkan mean, kita mungkin akan menganggap bahwa pelanggan kita berkantong tebal. Padahal, yang banyak terjual adalah yang harganya lebih murah.

Berikutnya adalah variabel a_total, ARRAY total penjualan, dengan nilai berikut:

```
[10000, 6000, 24000, 5000, 40000, 6000, 5000, 20000, 36000]
```

Nilai mean-nya (variabel t_mean) adalah 16.888,8889.

Artinya bagi kita:

- Ini adalah berapa banyak uang rata-rata yang ditinggalkan oleh satu pelanggan setiap kali datang.
- Dari sisi bisnis, walaupun harga rata-rata produk lebih murah, pelanggan juga membeli lebih dari 1 porsi.
- Setiap kali ada pelanggan, kita bisa berharap mendapatkan Rp 17.000.
- Angka Rp 16.888,8889 ini adalah key performance indicator yang penting. Jika ingin menaikkan omzet harian, pilihannya adalah mencari lebih banyak pelanggan, atau menaikkan nilai rata-rata ini. Misal dengan membuat paket kombinasi “kopi dan bakwan, diskon Rp 1.000.”
- Tentu saja masih dipengaruhi oleh outlier, yaitu Singkong keju seharga Rp 40.000. Tapi ada yang membeli 6 porsi bakwan seharga Rp 36.000, bukan? Rata-rata total Rp 16.888,8889 ini lebih “berbunyi” secara bisnis, dibanding rata-rata harga jual.

Median

Kembali ke variabel a_qty (ARRAY jumlah produk terjual) sebagai berikut:

```
[2, 1, 4, 1, 1, 1, 1, 4, 6]
```

Mediannya (variabel q_median) adalah 1.

Artinya bagi kita:

- Dengan mean, rata-rata jumlah terjual adalah 2,3333 porsi. Terasa semu karena ada pelanggan yang memborong 6 porsi sekaligus, misalnya. Berbeda dengan mean, median kebal terhadap nilai ekstrem. Dalam contoh ini, terlepas dari sedikit yang membeli dalam jumlah banyak, sebagian besar pelanggan sebenarnya hanya memesan 1 porsi per kunjungan.
- Tidak perlu membungkus terlebih dahulu per 2 sampai 3 porsi untuk mempercepat, karena penjualan didominasi oleh 1 porsi saja.

Untuk variabel a_price (ARRAY harga jual):

```
[5000, 6000, 6000, 5000, 40000, 6000, 5000, 5000, 6000]
```

Mediannya (variabel s_median) adalah 6.000.

Artinya bagi kita:

- Dengan mean, rata-rata harga jual adalah Rp 9.333,3333. Ini disebabkan karena harga jual Singkong keju yang Rp 40.000 tadi. Dengan median, bahkan jika kita naikkan harga Singkong keju menjadi 4 juta sekali pun (ada yang mau beli?), nilainya akan tetap Rp 6.000.
- Yang menggerakkan roda bisnis hari demi hari adalah produk dengan kisaran harga Rp 6.000.
- Jika ingin mengevaluasi daya beli, kita perlu gunakan yang Rp 6.000 ini, bukan yang Rp 9.333. Ini memastikan, kita tidak membuat kebijakan harga baru yang terlalu mahal.

Dan, untuk variabel a_total (ARRAY total penjualan):

```
[10000, 6000, 24000, 5000, 40000, 6000, 5000, 20000, 36000]
```

Mediannya (variabel t_median) adalah 10.000.

Artinya bagi kita:

- Bisa kita lihat bahwa dengan Rp 10.000 sebagai median, 50% dari total penjualan adalah Rp 10.000 ke bawah, dan 50% nya lagi adalah Rp 10.000 ke atas.
- Meskipun ada pelanggan yang menghabiskan Rp 40.000 sekali belanja, perlu diingat bahwa setengah dari aktivitas kasirnya dihabiskan untuk melayani penjualan senilai Rp 10.000 atau kurang.
- Jangan terlena dengan total penjualan yang besar.
- Kita mungkin perlu menyediakan uang kembalian dalam pecahan kecil, karena yang berbelanja kurang dari Rp 10.000 masih banyak.

Mode

Untuk variabel a_qty (ARRAY jumlah produk terjual) sebagai berikut:

```
[2, 1, 4, 1, 1, 1, 1, 4, 6]
```

Mode atau modusnya (variabel q_mode) adalah 1, sebanyak 5 kali. Contoh penggunaan fungsi mode:

```
> load_module("util")
> mode([2, 1, 4, 1, 1, 1, 1, 4, 6])
[1, 5]
```

Artinya bagi kita:

- Kemunculan 5 dari 9 kali yang tepat membeli 1 porsi artinya lebih dari 50 persen pelanggan yang berbelanja hanya akan membeli 1 porsi saja.
- Dengan demikian, bungkusan, kantong, atau piring saji dapat diprioritaskan untuk porsi sedikit tersebut.
- Dapat fokus pada kecepatan menyajikan.

Sementara, untuk variabel a_price (ARRAY harga jual):

```
[5000, 6000, 6000, 5000, 40000, 6000, 5000, 5000, 6000]
```

Modenya (variabel s_mode) adalah 5000, sebanyak 4 kali. Sesungguhnya, 6000 juga muncul 4 kali. Ini berarti, terdapat dua modus, atau bimodal. Hanya saja, di Singkong, fungsi mode hanya mengambil yang muncul pertama, yaitu 5000 (sebanyak 4 kali).

Artinya bagi kita:

- Zona nyaman pelanggan per produk adalah Rp 5.000. Bukan rata-rata Rp 9.333,3333.
- Apabila ingin membuat menu baru misal seharga Rp 24.000, kemungkin besar akan sepi peminat.
- Menjual barang di luar zona nyaman pelanggan adalah hal yang sangat beresiko.

Dan, untuk variabel a_total (ARRAY total penjualan):

```
[10000, 6000, 24000, 5000, 40000, 6000, 5000, 20000, 36000]
```

Modenya (variabel t_mode) adalah 6000, sebanyak 2 kali. Sama dengan sebelumnya, yaitu bimodal (5000

juga muncul 2 kali). Hanya saja, Singkong mengembalikan yang muncul pertama, yaitu 6000.

Artinya bagi kita:

- Kenyataannya, kasir paling banyak berhadapan dengan pelanggan yang hanya menyerahkan uang pas atau pecahan kecil untuk total belanja Rp 6.000 saja.

Tampaknya, warung makan kita didominasi oleh pesanan eceran porsi tunggal, untuk harga produk Rp 5.000, dan total penjualan Rp 6.000. Fokus layanan harus cepat, kemasan harus ramah untuk porsi tunggal, dan uang kembalian perlu disediakan.

Range

Untuk variabel a_qty (ARRAY jumlah produk terjual) sebagai berikut:

```
[2, 1, 4, 1, 1, 1, 1, 4, 6]
```

Nilai range-nya (variabel q_range) adalah 5.

Artinya bagi kita:

- Terdapat jarak yang masih wajar antara yang beli paling sedikit (1 porsi bakwan misalnya) dengan yang paling banyak (6 porsi bakwan). Rentang skalanya sejauh 5 porsi.
- Kita jadi tahu paling sibuk harus menyiapkan berapa porsi sekaligus. Beban kerja yang menyajikan juga tidak terlalu berbeda jauh.
- Tapi, sebenarnya, berdasarkan mode yang kita bahas sebelumnya, mayoritas pelanggan hanya

membeli 1 porsi. Range hanya peduli pada nilai yang ujung-ujung.

Untuk variabel a_price (ARRAY harga jual):
`[5000, 6000, 6000, 5000, 40000, 6000, 5000, 5000, 6000]`

Nilai range-nya (variabel s_range) adalah 35.000.

Artinya bagi kita:

- Berbeda dengan jumlah penjualan, nilai range 35.000 pada harga jual bukan sekedar selisih harga, melainkan karakteristik menu dan segmentasi pasar.
- Untuk warung makan kita, harga tertinggi (Singkong keju Rp 40.000 per porsi) bernilai 8 kali lipat dari harga terendah (Kopi Rp 5.000). Kita memiliki struktur menu yang tidak seimbang.
- Jika kopi kurang terjual misalnya, ruginya akan lebih kecil. Dibandingkan kalau Singkong keju yang tidak laku.
- Menu premium ini butuh strategi pemasaran khusus agar tidak menjadi pajangan yang hanya mengendapkan modal.

Untuk variabel a_total (ARRAY total penjualan):
`[10000, 6000, 24000, 5000, 40000, 6000, 5000, 20000, 36000]`

Nilai range-nya (variabel t_range) adalah 35.000.

Artinya bagi kita:

- Walaupun angkanya sama dengan range harga jual, ceritanya bisa berbeda.
- Ada pelanggan yang super hemat (Rp 5.000), dan ada yang kelas berat (Rp 40.000).
- Ada penjualan yang dipicu karena 1 barang premium. Namun, ada juga yang hampir menyentuh batas atas, yaitu Rp 36.000, karena pelanggan memborong 6 porsi produk yang lebih murah.
- Nilai range ini digerakkan oleh dua tipe pelanggan: pembeli produk premium dan pemborong produk yang lebih murah.

Standard Deviation

Untuk variabel a_qty (ARRAY jumlah produk terjual) sebagai berikut:

```
[2, 1, 4, 1, 1, 1, 1, 4, 6]
```

Nilai standard deviation (simpangan baku, variabel q_std) pada populasi adalah 1,7638.

Artinya bagi kita:

- Angka ini memberitahu seberapa konsisten perilaku belanja pelanggan di warung kita.
- Simpangan baku 1,7638 ini menunjukkan jumlah produk terjual mengumpul dan konsisten di sekitar nilai rata-ratanya (2,3333).
- Perilaku belanja relatif stabil. Kita tidak perlu khawatir ada kejutan harian orang yang belanja 50 porsi sekaligus.

- Batas bawah wajar 2,3333 – 1,7638 dapat dibulatkan menjadi 1 porsi.
- Batas atas wajar 2,3333 + 1,7638 dapat dibulatkan menjadi 4 porsi.
- Penjualan di luar itu, misal 6 porsi adalah sekedar bonus.

Untuk variabel a_price (ARRAY harga jual):
`[5000, 6000, 6000, 5000, 40000, 6000, 5000, 5000, 6000]`

Nilai standard deviation (simpangan baku, variabel s_std) pada populasi adalah 10.852,5471.

Artinya bagi kita:
- Nilai ini menunjukkan heterogenitas (keberagaman) yang sangat tinggi.
- Nilai ini bahkan lebih besar daripada nilai rata-ratanya itu sendiri, yaitu Rp 9.333,3333. Jika simpangan baku lebih besar dari rata-rata, data memiliki sebaran yang sangat lebar.
- Kita bisa melihat betapa ekstremnya harga Singkong keju menarik keluar jarak dari harga produk lainnya.
- Jarak harga antar produk terlalu renggang.
- Pendapatan warung akan sensitif terhadap kondisi apakah Singkong keju terjual atau tidak.
- Kita mungkin bisa memikirkan menu untuk menjembatani, misal makanan seharga Rp 15.000 atau Rp 20.000.

Untuk variabel a_total (ARRAY total penjualan):
`[10000, 6000, 24000, 5000, 40000, 6000, 5000, 20000, 36000]`

Nilai standard deviation (simpangan baku, variabel t_std) pada populasi adalah 13.025,1419.

Artinya bagi kita:

- Nilai ini mencerminkan tingkat stabilitas arus kas kasir.
- Nilai ini tergolong sangat besar karena hampir mendekati nilai rata-ratanya sendiri (Rp 16.888,8889).
- Omzet yang masuk dari satu nota ke nota lain sangat fluktuatif. Kasir bisa saja melayani transaksi Rp 5.000, kemudian melompat ke Rp 40.000.
- Kita bisa terapkan batas bawah wajar dan batas atas wajar, seperti pada jumlah pembelian.

Mari kembali ke kode program kita. Mungkin ada baiknya, apabila kita tampilkan total untuk kolom jumlah, harga, dan total penjualan, lengkap dengan terbilangnya. Baris 58 sampai 61 mencontohkan fungsi sum (untuk menjumlahkan keseluruhan elemen dalam ARRAY) dan words_id (untuk terbilang dalam Bahasa Indonesia, dipanggil dengan argumen berupa STRING). Pastikanlah kita tambahkan ke ARRAY r_a dengan ARRAY sesuai jumlah dan posisi kolom yang kita inginkan dalam tabel.

```
r_a + ["Jumlah Total", "", ng(sum(a_qty)), "", ng(sum(a_total))]
r_a + ["Terbilang", "", words_id(string(sum(a_qty))), "",
    words_id(string(sum(a_total)))]
```

Mulai baris 62 sampai 136, kode program kita sepenuhnya fokus pada sisi GUI. Selain buku referensi bahasa Singkong (Mengenal dan Menggunakan Bahasa Pemrograman Singkong), terdapat dua buku contoh lain untuk GUI (Contoh dan Penjelasan Bahasa Singkong: Dasar-dasar Aplikasi GUI; Contoh dan Penjelasan Bahasa Singkong: Mahir Bekerja dengan GUI), apabila Anda tertarik lebih lanjut dengan GUI di Singkong. Untuk buku ini, pembahasan dari sisi GUI akan dilakukan secara lebih ringkas.

Di baris 62 sampai 64, kita memanggil reset (diperlukan untuk aplikasi GUI) dan menentukan titel frame:

```
reset()
title("Dashboard")
```

Di baris 65 sampai 73, kita membuat COMPONENT table (fungsi component, variabel t_a) dan menentukan alignment kolom (dimulai dari 0), dimana jumlah (indeks 2), harga (indeks 3), dan total (indeks 4) dibuat rata kanan (fungsi table_right). Kemudian, kita tentukan isi tabel sesuai variabel r_a (dengan fungsi config). Penulis membuat sebuah grid (variabel g_a) untuk menampung tabel tersebut (ukuran maksimal dalam grid).

```
var t_a_cols = "Waktu,Produk,Jumlah,Harga,Total"
var t_a = component("table", t_a_cols, true)
table_right(t_a, 2)
table_right(t_a, 3)
table_right(t_a, 4)
config(t_a, "contents", r_a)
var g_a = component("grid", "")
grid_add(g_a, t_a, 0, 0, 1, 1, 1, 1, 3, 0)
```

Di sisi kiri bawah frame kita adalah sebuah tabel lain (variabel t_b), berisikan hasil analisis kita. Variabel r_b adalah isi tabel. Berikut adalah baris 74 sampai 103. Jumlah barisnya banyak karena kita membuat ARRAY r_b secara manual. Tapi pada dasarnya, caranya sama saja ketika kita menyiapkan tabel t_a sebelumnya.

```
var t_b_cols = "Metrik,Nilai"
var t_b = component("table", t_b_cols, true)
table_right(t_b, 1)
var r_b = [
    ["NILAI TRANSAKSI", ""],
    ["Rata-rata nilai transaksi", ng(t_mean)],
    ["Nilai transaksi paling mewakili", ng(t_median)],
    ["Nilai transaksi paling sering muncul",
        ng(t_mode[0]) + " (" + t_mode[1] + " kali)"],
    ["Rentang nilai transaksi", ng(t_range)],
    ["Variasi nilai transaksi (std dev)", ng(t_std)],
    ["", ""],
    ["HARGA PRODUK", ""],
    ["Rata-rata harga produk", ng(s_mean)],
    ["Harga produk paling mewakili", ng(s_median)],
    ["Harga produk paling sering muncul",
        ng(s_mode[0]) + " (" + s_mode[1] + " kali)"],
    ["Rentang harga produk", ng(s_range)],
    ["Variasi harga produk (std dev)", ng(s_std)],
    ["", ""],
    ["JUMLAH PEMBELIAN", ""],
    ["Rata-rata jumlah pembelian", ng(q_mean)],
    ["Jumlah pembelian paling mewakili", ng(q_median)],
    ["Jumlah pembelian paling sering muncul",
        ng(q_mode[0]) + " (" + q_mode[1] + " kali)"],
    ["Rentang jumlah pembelian", ng(q_range)],
    ["Variasi jumlah pembelian (std dev)", ng(q_std)]
]
config(t_b, "contents", r_b)
```

Di baris 104 sampai 115, kita membuat COMPONENT barchart (variabel c_a) untuk menampilkan perbandingan jumlah penjualan. Kita mengkonfigurasi beberapa aspek barchart tersebut dengan fungsi config yang sama.

```
var c_a = component("barchart", "")
```

```
config(c_a, "foreground", "black")
config(c_a, "background", "white")
config(c_a, "font", ["monospaced", 1, 14])
config(c_a, "text", "Jumlah Terjual")
var r_c_a = []
each(sort_string(keys(sold)), fn(e, i) {
    var q = sold[e]
    r_c_a + [q, e + " " + q, random(colors)]
})
config(c_a, "contents", r_c_a)
```

Yang agak berbeda barangkali adalah each untuk key dari variabel sold. Tujuannya adalah membuat ARRAY [nilai, label, warna] untuk setiap batang dalam barchart. Nantinya, kita tambahkan dalam ARRAY dari ARRAY (variabel r_c_a) yang akan menjadi isi dari barchart. Dengan variabel sold, tentu saja nilainya adalah jumlah. Sementara, warnanya, kita pilih secara acak dari COLORS (dengan fungsi random).

Fungsi sort_string digunakan untuk mengurutkan ARRAY dari STRING. Jadi, “bakwan” akan tampil duluan dibandingkan “singkong keju” dalam barchart.

Baris 116 sampai 126 adalah barchart lainnya (variabel c_b), untuk menampilkan dari sisi omzet penjualan. Baris-baris kodenya mirip dengan barchart sebelumnya. Hanya saja, dasar dari isi barchart adalah variabel sold_total.

```
var c_b = component("barchart", "")
config(c_b, "foreground", "black")
config(c_b, "background", "white")
config(c_b, "font", ["monospaced", 1, 14])
config(c_b, "text", "Omzet Penjualan")
var r_c_b = []
each(sort_string(keys(sold_total)), fn(e, i) {
```

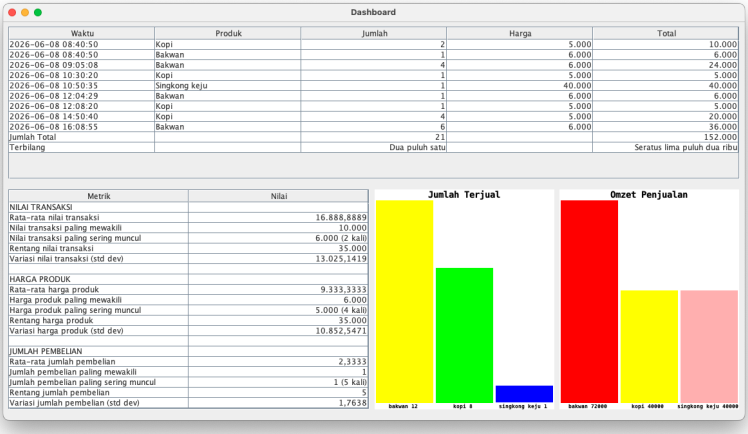
```
    var q = sold_total[e]
    r_c_b + [q, e + " " + q, random(colors)]
})
config(c_b, "contents", r_c_b)
```

Di baris 127 sampai 131, kita buat grid lain (variabel g_b), dimana kita tambahkan tabel hasil analisis (variabel t_b, lebar 50%, tinggi maksimum), dan kedua barchart (variabel c_a dan c_b, lebar masing-masing 25%, tinggi maksimum).

```
var g_b = component("grid", "")
grid_add(g_b, t_b, 0, 0, 1, 1, 0.5, 1, 3, 0)
grid_add(g_b, c_a, 1, 0, 1, 1, 0.25, 1, 3, 0)
grid_add(g_b, c_b, 2, 0, 1, 1, 0.25, 1, 3, 0)
```

Lima baris terakhir, yaitu 132 sampai 136, kita buat sebuah grid lagi (variabel g), yang akan menampung dua grid yang kita buat sebelumnya (variabel g_a tinggi 40% dan variabel g_b tinggi 60%). Lalu, kita tambahkan grid g ke frame (dengan fungsi add), dan tampilkan framenya (dengan fungsi show).

Program kita pun selesai. Simpanlah dengan tombol Save, dan jalankanlah dengan fungsi Run. Berikut adalah contoh tampilan program kita. Warna batang pada barchart di komputer Anda tentunya dapat berbeda, karena fungsi random.


Dashboard
Waktu Produk Jumlah Harga Total
2026-06-08 08:40:50 Kopi 2 5.000 10.000
2026-06-08 08:40:50 Bakwan 1 6.000 6.000
2026-06-08 09:05:08 Bakwan 4 6.000 24.000
2026-06-08 10:30:20 Kopi 1 5.000 5.000
2026-06-08 10:50:35 Singkong keju 1 40.000 40.000
2026-06-08 12:04:29 Bakwan 1 6.000 6.000
2026-06-08 12:08:20 Kopi 1 5.000 5.000
2026-06-08 14:50:40 Kopi 4 5.000 20.000
2026-06-08 16:08:55 Bakwan 6 6.000 36.000
Jumlah Total 21 152.000
Terbilang Dua puluh satu Seratus lima puluh dua ribu
Metrik Nilai
NILAI TRANSAKSI
Rata-rata nilai transaksi 16.888,8889
Nilai transaksi paling mewakili 10.000
Nilai transaksi paling sering muncul 6.000 (2 kali)
Rentang nilai transaksi 35.000
Variasi nilai transaksi (std dev) 13.025,1419
HARGA PRODUK
Rata-rata harga produk 9.333,3333
Harga produk paling mewakili 6.000
Harga produk paling sering muncul 5.000 (4 kali)
Rentang harga produk 35.000
Variasi harga produk (std dev) 10.852,5471
JUMLAH PEMBELIAN
Rata-rata jumlah pembelian 2,3333
Jumlah pembelian paling mewakili 1
Jumlah pembelian paling sering muncul 1 (5 kali)
Rentang jumlah pembelian 5
Variasi jumlah pembelian (std dev) 1,7638
Jumlah Terjual
bakwan 12
kopi 8
singkong keju 1
Omzet Penjualan
bakwan 72000
kopi 40000
singkong keju 40000

Laporan, Sumber Data Lain, dan Program Entri Data

Laporan

Mari kita buat sebuah program GUI lain, yang akan menampilkan data penjualan yang sudah clean, dan menyediakan sebuah tombol, yang ketika di klik, akan menghasilkan sebuah laporan dalam sebuah file teks, lengkap dengan tabel.

Berikut adalah contoh laporan yang dimaksud:

```
==========================================================================
|       Waktu        |       Produk       | Jumlah | Harga  |  Total   |
==========================================================================
|2026-06-08 08:40:50 |Kopi                |       2|   5.000|    10.000|
--------------------------------------------------------------------------
|2026-06-08 08:40:50 |Bakwan              |       1|   6.000|     6.000|
--------------------------------------------------------------------------
|2026-06-08 09:05:08 |Bakwan              |       4|   6.000|    24.000|
--------------------------------------------------------------------------
|2026-06-08 10:30:20 |Kopi                |       1|   5.000|     5.000|
--------------------------------------------------------------------------
|2026-06-08 10:50:35 |Singkong keju       |       1|  40.000|    40.000|
--------------------------------------------------------------------------
|2026-06-08 12:04:29 |Bakwan              |       1|   6.000|     6.000|
--------------------------------------------------------------------------
|2026-06-08 12:08:20 |Kopi                |       1|   5.000|     5.000|
--------------------------------------------------------------------------
|2026-06-08 14:50:40 |Kopi                |       4|   5.000|    20.000|
--------------------------------------------------------------------------
|2026-06-08 16:08:55 |Bakwan              |       6|   6.000|    36.000|
--------------------------------------------------------------------------
|Jumlah Total        |                    |      21|        |   152.000|
==========================================================================

Jumlah terjual : Dua puluh satu
Omzet penjualan: Seratus lima puluh dua ribu
Dihasilkan pada: 2026-06-21 21:00:50
```

Lumayan bagus, bukan? Bilangan tetap diformat dengan pemisah ribuan dan kolomnya tetap disajikan rata kanan. Penulis yakin sekali bahwa untuk kode program GUI nya, hanya sangat sedikit yang perlu dibahas, karena untuk menampilkan tabel, kita sudah lakukan sebelumnya.

Program kita hanya terdiri dari 60 baris. Simpanlah sebagai x-table-text.singkong. Berikut adalah 43 baris pertama, yang sama sekali tidak perlu dibahas lebih lanjut—Anda sudah membuat yang lebih sulit.

```
load_module("csv")
load_module("util")
load_module("ui_util")

var a = csv_from_string(",", read("data_clean.csv"))
var r = []

var a_qty = []
var a_total = []

var ng = number_group_p_c
each(a, fn(e, i) {
    var p = e[1]
    var q = number(e[2])
    var s = number(e[3])
    var t = q * s
    a_qty + q
    a_total + t
    r + [
        e[0],
        upper(e[1], true),
        ng(q),
        ng(s),
        ng(t)
    ]
})

r + ["Jumlah Total", "", ng(sum(a_qty)), "", ng(sum(a_total))]

reset()
title("Report")

var t_cols = "Waktu,Produk,Jumlah,Harga,Total"
var t = component("table", t_cols, true)
```

```
table_right(t, 2)
table_right(t, 3)
table_right(t, 4)
config(t, "contents", r)
var b = component("button", "Simpan")
add(t)
add_s(b)
show()
```

Yang berbeda hanyalah:

- penambahan COMPONENT table (variabel t) dilakukan dengan fungsi add, tanpa lewat grid atau panel. Langsung pada frame.
- Pembuatan sebuah COMPONENT button (variabel b), yang ditambahkan pada sisi bawah (south) frame dengan fungsi add_s.
- Kita load modul tambahan, yaitu ui_util untuk kebutuhan menghasilkan tabel teks berdasarkan COMPONENT table.

Yang perlu kita perhatikan adalah baris 44 sampai 60 berikut:

```
event(b, fn() {
    var f = "report.txt"
    var h = [20, 20, 8, 8, 10]
    var c = table_to_text(t, h)
    var c = c + newline() +
        "Jumlah terjual : " + words_id(string(sum(a_qty)))
    var c = c + newline() +
        "Omzet penjualan: " + words_id(string(sum(a_total)))
    var c = c + newline() +
        "Dihasilkan pada: " + @
    var w = write(f, c)
    if (w == true) {
        message(f + " berhasil disimpan")
    } else {
        message(f + " gagal disimpan")
    }
})
```

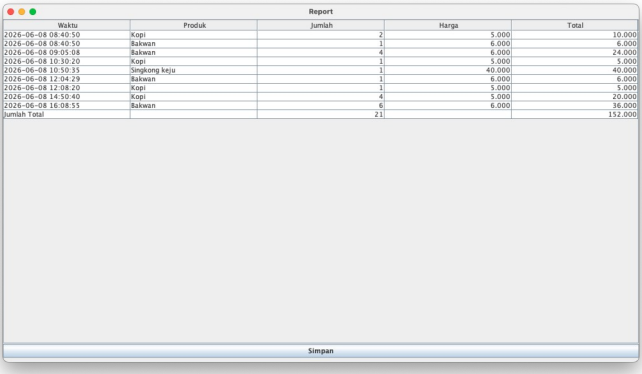
Apabila tombol b ditekan:

```
event(b, fn() {
})
```

Kita tangani dengan sebuah fungsi tanpa nama (fn () di baris 44), yang akan:

- Menentukan kolom dan lebarnya dalam ARRAY. Perhatikanlah variabel h kita.
- Kita ubah tabel ke teks dengan fungsi table_to_text dari modul ui_util, dan simpan sebagai variabel c (STRING).
- Variabel c tersebut kemudian kita tambahkan beberapa informasi tambahan, dipisahkan baris kosong dengan fungsi newline.
- Kita tulis ke file “report.txt” (variabel f) dengan fungsi write, dan tampilkan pesan berdasarkan apakah write berhasil atau tidak. Apabila nama file ingin agar dapat dipilih oleh user, gunakanlah fungsi save untuk menampilkan dialog.

Save dan Run lah program kita. Berikut contohnya:



Sumber data lain

Di buku ini, kita mencontohkan penggunaan format CSV. Tapi, pastinya, data bisa disimpan pada berbagai format atau sistem lain. Beberapa contohnya:

- Apabila menggunakan file yang dibaca/tulis secara langsung, seperti ketika menggunakan spreadsheet, kita bekerja dengan format dokumen tersendiri (misal dalam file dengan ekstensi nama file .xlsx, .ods, atau .numbers).
- Berbagai aplikasi mungkin menyimpan datanya dalam sistem database relasional, di mana kita bisa mengakses file atau direktorinya secara langsung. Misal dengan SQLite atau Apache Derby.
- Pada sistem database tertentu, kita mungkin mengaksesnya lewat protokol tersendiri. Misal ketika kita menggunakan PostgreSQL.
- Data mungkin kita dapatkan lewat web service. Bisa jadi filenya memang sudah ada di web server, atau dihasilkan secara dinamis setelah melewati otentikasi dan otorisasi.

Ayo, kita baca dari data_clean.csv dan simpan ke sumber data lain.

Kita mulai dengan Apache Derby (secara embedded). Tidak diperlukan konfigurasi sistem apapun. Simpanlah kode program berikut sebagai x-csv-db.singkong. Hanya dalam 36 baris.

Berikut adalah 16 baris pertama kodenya, dimana kita melakukan koneksi database (dibuat otomatis apabila belum ada), dan membuat tabelnya.

```
load_module("csv")
load_module("db_util")

var d = db_connect_embed("data")
if (d != null) {
    db_create_table_embed(d, "sales",
        [
            ["id", "id."],
            ["waktu", "timestamp."],
            ["produk", "varchar"],
            ["jumlah", "integer"],
            ["harga", "decimal"],
            ["total", "decimal"]
        ]
    )
```

Kita perlu load modul csv karena akan membaca dari data_clean.csv, dan modul db_util karena akan menggunakan beberapa fungsi bantu untuk bekerja dengan sistem database relasional.

Kita akan mencoba melakukan koneksi (dan mencoba membuat databasenya):

```
var d = db_connect_embed("data")
```

Apabila berhasil, d akan bertipe DATABASE, dan tidak null. Kita pun membuat tabelnya (dengan nama “sales”):

```
    db_create_table_embed(d, "sales",
        [
            ["id", "id."],
            ["waktu", "timestamp."],
            ["produk", "varchar"],
            ["jumlah", "integer"],
            ["harga", "decimal"],
            ["total", "decimal"]
        ]
    )
```

Apabila diperlukan saja, Anda mungkin tertarik membaca buku: Contoh dan Penjelasan Bahasa Singkong: Bekerja dengan Database Relasional. Terkait titik pada “id.” dan “timestamp.”, dimaksudkan sebagai memiliki nilai default tertentu. Kita menentukan tipe yang sesuai supaya tidak perlu terus menerus mengkonversi dari STRING.

Berikut adalah baris 17 sampai 31:

```
var a = csv_from_string(",", read("data_clean.csv"))
each(a, fn(e, i) {
    var q = number(e[2])
    var s = number(e[3])
    db_insert(d, "sales",
        {
            "waktu": e[0],
            "produk": e[1],
            "jumlah": q,
            "harga": s,
            "total": q * s
        }
    )
})
```

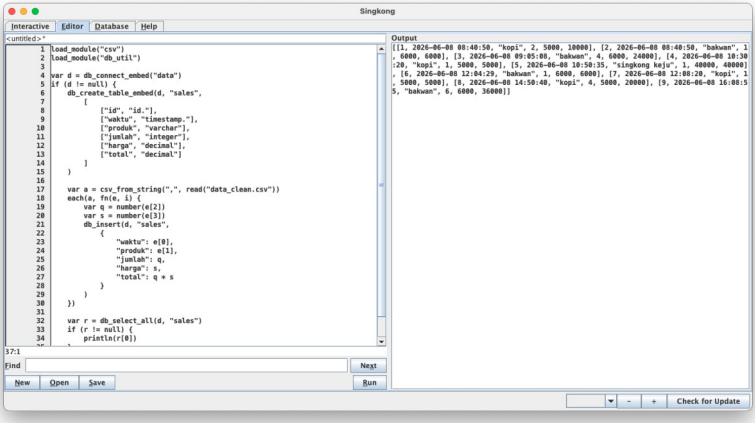
Membaca dari CSV dan perulangan tentunya sudah kita lakukan sebelumnya. Untuk setiap baris yang berhasil dibaca, kita simpan ke tabel tersebut dengan fungsi db_insert.

```
    db_insert(d, "sales",
        {
            "waktu": e[0],
            "produk": e[1],
            "jumlah": q,
            "harga": s,
            "total": q * s
        }
    )
```

Setelah selesai, untuk menguji, kita baca semua isi tabel, dan apabila query berhasil, kita tampilkan. Berikut adalah baris 32 sampai 36.

```
    var r = db_select_all(d, "sales")
    if (r != null) {
        println(r[0])
    }
}
```

Kliklah tombol Save dan kemudian Run. Berikut adalah contoh tampilan programnya.



Nantinya, di contoh program entri data, kita akan membaca dari dan menulis ke tabel yang sama.

Contoh berikutnya adalah dari CSV, kita konversi ke JSON, yang dapat di-upload ke web server misalnya. Penulis telah menyediakan contoh di repo GitHub, yang kemudian dapat kita baca lewat HTTP.

Untuk mengkonversi dari data_clean.csv ke data.json dalam format serupa (ARRAY dari ARRAY), kita hanya

membutuhkan beberapa baris berikut. Simpanlah sebagai x-csv-json.singkong.

```
load_module("csv")
load_module("json")

var a = csv_from_string(",", read("data.csv"))
write("data.json", json_string(a))
```

Jalankanlah programnya. Apabila berhasil, true akan ditampilkan, dan file data.json dengan isi berikut akan ditemukan di direktori aktif.

```
[["waktu", "produk", "jumlah", "harga"], ["2026-06-08
08:40:50", "Kopi", "2", "5000"], ["2026-06-08 08:40:50",
"bakwan", "1", "6000"], ["2026-06-08 09:05:08",
"bakwan", "4", "Rp 6000"], ["2026-06-08 10:30:20",
"kopi", "", "5000"], ["2026-06-08 10:50:35", "Sinkong
Keju", "1", "40000"], ["08/06/2026 12:04:29", "bakwan",
"1", "6000"], ["2026-06-08 12:08:20", "kpi", "1",
"5000"], ["2026-06-08 14:50:40", "kopi", "4", "5000"],
["2026-06-08 16:08:55", "Bak wan", "6", "6000"], ["2026-
06-08 16:20:58", "bakwan", "3"]]
```

Untuk file yang telah penulis siapkan di repo GitHub, kita dapat mengaksesnya dengan contoh berikut. Simpanlah sebagai x-load_data_http.singkong.

```
load_module("json")

var u =
"https://raw.githubusercontent.com/nopri/example/refs/h
eads/main/data_analysis/data.json"
var r = http_get(u)
if (r != null) {
    var d = r[2]
    var a = json_parse(d)
    println(a)
}
```

Jalankanlah program tersebut (Anda akan membutuhkan koneksi Internet), dan apabila berhasil, isi file JSON tersebut akan ditampilkan.

Program entri data

Mari sejenak kita lihat ulang kenapa data.csv tidak bersih. Hanya karena input manual saja kan? Kalau ada program entri datanya, pastilah kita tidak perlu sibuk membuat fungsi-fungsi untuk membersihkan data.

Baiklah, mari kita buat program sederhana untuk kebutuhan tersebut. Simpanlah sebagai x-data-entry-db.singkong. Program ini membutuhkan 100 baris kode. Tapi sebagian besar tidak perlu dibahas lagi. Sekali lagi, Anda telah membuat yang lebih rumit.

Berikut adalah 12 baris pertamanya:

```
load_module("csv")
load_module("util")
load_module("db_util")

var P_BAKWAN = "bakwan/6000"
var P_KOPI = "kopi/5000"
var P_SING_K = "singkong keju/40000"
var PRODUCTS = [P_BAKWAN, P_KOPI, P_SING_K]

var d = db_connect_embed("data")
var ng = number_group_p_c
```

Yang berbeda adalah:

- Kita load modul db_util untuk bekerja dengan sistem database relasional. Kemudian, kita buat/koneksi ke database embedded dengan

nama “data”. Ketika dijalankan nanti, direktori tersebut akan ada di direktori aktif (apabila berhasil).

- Perhatikanlah bahwa produk kita kini memiliki informasi harga, dipisahkan sebuah “/”. Nantinya, kita akan split.
- Anda pastilah menebak: kalau begitu, kita tidak baca produk dan harganya dari tabel database! Benar sekali. Desain database kita tidak ternormalisasi (denormalized). Untuk buku ini, fokus kita adalah contoh analisis data sederhana, tanpa harus masuk ke pembahasan tentang sistem database relasional.

Baris 13 sampai 50 berikut adalah fungsi load_data, yang akan membaca isi tabel database. Fungsi ini akan kita panggil ketika program dijalankan, dan ketika penjualan baru berhasil disimpan. Supaya bisa kita tampilkan data terbaru.

```
var load_data = fn() {
    var a = db_query_single(d, "
        select waktu,produk,jumlah,harga,total
        from sales
        order by waktu asc",
        []
    )
    if (a == null) {
        var a = [[]]
    }
    var a = a[0]
    var r = []

    var a_qty = []
    var a_price = []
    var a_total = []

    each(a, fn(e, i) {
        var p = e[1]
        var q = e[2]
        var s = e[3]
        var t = e[4]
        a_qty + q
```

```
        a_total + t
        r + [
            e[0],
            upper(e[1], true),
            ng(q),
            ng(s),
            ng(t)
        ]
    })
    r + ["Jumlah Total", "", ng(sum(a_qty)), "", ng(sum(a_total))]
    r + ["Terbilang", "", words_id(string(sum(a_qty))), "",
        words_id(string(sum(a_total)))]
    return r
}
```

Lihatlah. Yang berbeda hanyalah ketika membaca isi tabel sales dengan db_query_single dan memeriksa apakah hasil query mengembalikan null (berguna ketika query kita keliru atau kendala komunikasi dengan sistem databasenya). Kalau tidak null, kita ambil elemen pertama dalam ARRAY. Fungsi db_query_single dari modul db_util akan memanggil fungsi query, dan fungsi ini bisa menerima menjalankan perintah sekaligus (sebagai satu transaksi), yang mana, hasilnya juga akan berupa beberapa elemen dalam ARRAY.

Baris 51 sampai 59 berikut adalah fungsi save_data, dimana data yang dientri akan disimpan dalam tabel database. Kita gunakan fungsi db_insert dari modul db_util untuk insert ke tabel sales.

```
var save_data = fn(p, s, q) {
    db_insert(d, "sales", {
        "produk": p,
        "jumlah": q,
        "harga": s,
        "total": q * s
    })
}
```

Baris 60 sampai 65 adalah fungsi set_contents. Fungsi ini akan memanggil load_data dan hasilnya akan ditampilkan dalam tabel. Lalu, kita scroll tabelnya ke baris paling bawah dengan table_scroll.

```
var set_contents = fn(t) {
    var res = load_data()
    config(t, "contents", res)
    table_scroll(t, len(res) - 1)
}
```

Baris 66 sampai 92 berikut tidak butuh banyak penjelasan. Yang beda hanya ketika kita membuat COMPONENT combobox (untuk nama produk, dengan isi adalah variabel PRODUCTS) dan spin (untuk jumlah, maksimum 200, minimal 1, default 1, step 1).

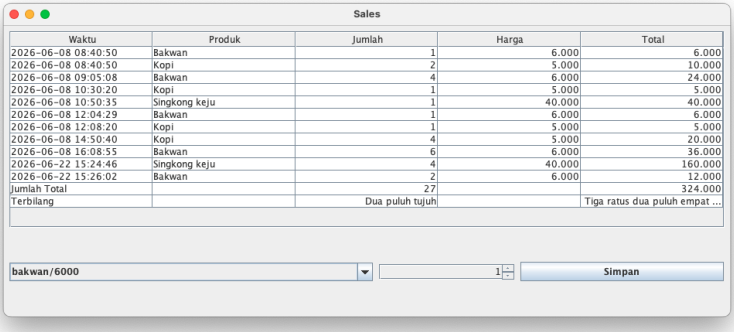
```
reset()
title("Sales")

var t_cols = "Waktu,Produk,Jumlah,Harga,Total"
var t = component("table", t_cols, true)
table_right(t, 2)
table_right(t, 3)
table_right(t, 4)
set_contents(t)
var g_a = component("grid", "")
grid_add(g_a, t, 0, 0, 1, 1, 1, 1, 3, 0)

var p = component("combobox", "")
config(p, "contents", PRODUCTS)
var q = component("spin", "1,1,200,1")
var g_b = component("grid", "")
var b = component("button", "Simpan")
grid_add(g_b, p, 0, 0, 1, 1, 0.5, 1, 1, 0)
grid_add(g_b, q, 1, 0, 1, 1, 0.2, 1, 1, 0)
grid_add(g_b, b, 2, 0, 1, 1, 0.3, 1, 1, 0)

var g = component("grid", "")
grid_add(g, g_a, 0, 0, 1, 1, 1, 0.8, 3, 0)
grid_add(g, g_b, 0, 1, 1, 1, 1, 0.2, 3, 0)
```

```
add(g)
show()
```

Baris 93 sampai 100 adalah ketika kita menangani tombol Simpan, lakukan konfirmasi (dengan confirm, kalau berhasil, mengembalikan “OK”), dapatkan produk/jumlah pesanan, split nama produk/harganya, panggil save_data, dan set_contents.

```
event(b, fn() {
    if (confirm("Ingin menyimpan?") == "OK") {
        var _p = split(get(p, "text"), "/")
        var _q = get(q, "contents")
        save_data(_p[0], number(_p[1]), _q)
        set_contents(t)
    }
})
```

Program kita pun selesai. Tekanlah Save dan Run. Berikut adalah contoh hasilnya (ukuran frame telah diubah secara manual), lengkap dengan tambahan data penjualan.



Bekerja dengan Spreadsheet

Dr. Maria Seraphina Astriani

Data dapat menjadi tidak bersih (atau sering disebut *dirty/unclean data*) terjadi karena proses pengumpulan, penginputan, dan penyimpanan data melibatkan banyak faktor.

Pada pemaparan pembahasan sebelumnya, kita dapat melihat bahwa salah satu penyebabnya dapat disebabkan oleh tingkah laku user, migrasi, masalah teknis, dan lainnya.

Mari kita lihat secara detail masalah yang ada serta bagaimana teknik yang dapat digunakan untuk membersihkan data dengan menggunakan Spreadsheet.

Tingkah Laku Pengguna (User Behavior)

Data yang Dikosongkan

Pengguna sengaja mengosongkan data yang tidak wajib diisi, sehingga menghasilkan nilai kosong (missing values). Contoh:

	A	B
1	**Nama**	**Umur**
2	Anna	30
3	Roni	50
4	Silvia	
5	Zaky	40

Penanganan data dengan nilai kosong cukup *tricky* karena nilai kosong tersebut bukan kesalahan data, melainkan bagian dari informasi dan cara membersihkannya bergantung pada tujuan analisis yang diinginkan.

Apabila tujuannya adalah “ingin menjaga data agar tetap mencerminkan jawaban responden”, nilai kosong ini dapat dibiarkan kosong.

Namun jika kolom tersebut sangat penting untuk analisis, kita dapat menghapus seluruh baris yang memiliki nilai kosong dengan cara melakukan hal berikut:

- Pilih Data → Create a filter
- Filter kolom tersebut
- Pilih Blanks
- Hapus baris yang muncul

Sehingga hasil pada Spreadsheet akan menjadi seperti ini:

	A	B
1	**Nama**	**Umur**
2	Anna	30
3	Roni	50
4	Zaky	40

Jika analisis memerlukan data lengkap, maka nilai kosong dapat diisi dengan nilai pengganti (imputasi).

Apabila data kategorik, dapat diisi dengan kategori yang paling sering muncul (modus).

Apabila data berupa data numerik (seperti pada contoh ini yaitu umur), nilai kosong dapat diisi dengan rata-rata atau median. Sehingga umur “Silvia” dapat diisi dengan nilai 40 (dihitung dari ((30 + 50 + 40) / 3) dan hasil yang diperoleh akan seperti berikut:

	A	B
1	**Nama**	**Umur**
2	Anna	30
3	Roni	50
4	Silvia	40
5	Zaky	40

Input Asal-Asalan

Kejadian input asal-asalan seperti ini umumnya sering terjadi karena Pengguna tidak mengetahui data yang benar dan untuk mempercepat proses input data. Contoh:

	A	B
1	**Nama**	**Telepon**
2	Anna	62888192733
3	Roni	1233333
4	Silvia	628889372872
5	Zaky	123123

Pada contoh diatas dapat dilihat bahwa nomor “Telepon” Roni dan Zaky diisi secara asal-asalan.

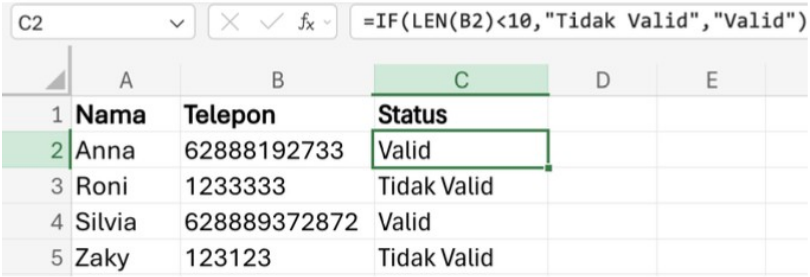
Pada Spreadsheet, apabila ingin melakukan menandai dengan membuat kolom baru (pada kolom C) dan melakukan validasi bahwa panjang "Telepon" tidak boleh kurang dari 10 digit dapat menggunakan rumus berikut pada Google Sheet:

```
=IF(LEN(B2)<10,"Tidak Valid","Valid")
```

Atau rumus berikut pada Excel:

```
=IF(LEN(B2)<10;"Tidak Valid";"Valid")
```

Hasil yang didapatkan akan seperti berikut:



	A	B	C	D	E
1	**Nama**	**Telepon**	**Status**		
2	Anna	62888192733	Valid		
3	Roni	1233333	Tidak Valid		
4	Silvia	628889372872	Valid		
5	Zaky	123123	Tidak Valid		

Cara lain untuk menangani data input asal-asalan tersebut dapat dilakukan dengan cara menghapus seluruh baris yang memiliki nilai kosong sehingga data yang dihasilkan akan seperti berikut:

	A	B
1	**Nama**	**Telepon**
2	Anna	62888192733
3	Silvia	628889372872

Apabila terdapat kasus kolom "Telepon" kosong sebanyak 80%, maka seluruh kolom tersebut dihapus dari analisis karena tidak cukup kuat untuk digunakan.

Kesalahan Manusia (Human Error)

Format yang tidak konsisten

Format yang tidak konsisten ini sering kali terjadi pada saat Pengguna menuliskan tanggal dengan format yang berbeda-beda pada satu kolom. Sebagai contoh, format tanggal DD-MM-YYYY dapat bercampur dengan DD-MM-YY atau DD/MM/YY dan lainnya).

Contoh di bawah ini merupakan data Tanggal Lahir (“TglLahir”) dengan format yang berlainan.

	A	B
1	**Nama**	**TglLahir**
2	Anna	5/31/2010
3	Roni	8/16/2013
4	Silvia	27-06-2013
5	Zaky	3/24/2011

Tanggal Lahir dengan tanda garis miring (/) mempunyai format mm/dd/yyyy. Sedangkan Tanggal Lahir dengan tanda minus (-) mempunyai format dd-mm-yyyy. Mari kita coba samakan format tersebut dengan format dd-mm-yyyy.

Berikut ini adalah formula yang dapat digunakan:

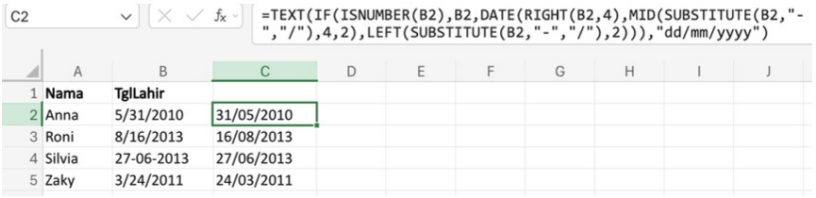
```
=TEXT(IF(ISNUMBER(B2),B2,DATE(RIGHT(B2,4),MID(SUBSTITUTE(
B2,"-","/"),4,2),LEFT(SUBSTITUTE(B2,"-","/"),2))),"dd/
mm/yyyy")
```

Rumus ini memastikan nilai pada cell B2 selalu ditampilkan dalam format dd/mm/yyyy, baik jika data awal sudah berupa tanggal maupun masih berupa teks dengan format dd-mm-yyyy.

Penjelasan:

- ISNUMBER(B2) berfungsi untuk mengecek apakah isi sel B2 sudah berupa tanggal yang dikenali Excel/Google Sheets.
- IF(ISNUMBER(B2), B2, ...) dapat digunakan untuk pengecekan. Apabila data sudah berupa tanggal, gunakan langsung. Jika masih berupa teks, maka akan diubah menjadi tanggal.
- SUBSTITUTE(B2,"-","/") berfungsi untuk mengganti tanda - menjadi /
- RIGHT(B2,4) digunakan untuk mengambil tahun (4 digit terakhir).
- MID(...,4,2) berfungsi untuk mengambil 2 digit di tengah.
- LEFT(...,2) mempunyai kemampuan untuk mengambil 2 digit pertama.
- DATE(tahun,bulan,hari) digunakan untuk menyusun kembali hari, bulan, dan tahun menjadi format tanggal yang dikenali Excel/Google Sheets.
- TEXT(...,"dd/mm/yyyy") berfungsi untuk menampilkan hasil akhir dalam format dd/mm/yyyy.

Berikut merupakan hasil akhir setelah berhasil mengubah Tanggal Lahir dengan menggunakan formula diatas (nomor cell mohon dapat disesuaikan dengan kebutuhan).

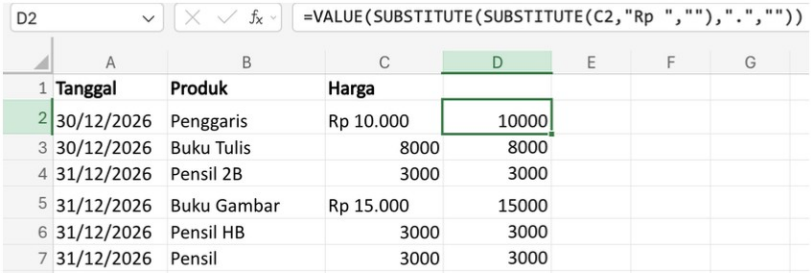


	A	B	C	D	E	F	G	H	I	J
1	**Nama**	**TglLahir**								
2	Anna	5/31/2010	31/05/2010							
3	Roni	8/16/2013	16/08/2013							
4	Silvia	27-06-2013	27/06/2013							
5	Zaky	3/24/2011	24/03/2011							

Contoh lainnya format tidak konsisten yang terjadi adalah data “Harga” produk.

	A	B	C
1	**Tanggal**	**Produk**	**Harga**
2	30/12/2026	Penggaris	Rp 10.000
3	30/12/2026	Buku Tulis	8000
4	31/12/2026	Pensil 2B	3000
5	31/12/2026	Buku Gambar	Rp 15.000
6	31/12/2026	Pensil HB	3000
7	31/12/2026	Pensil	3000

Kolom "Harga" pada data penjualan terbaca sebagai teks karena ada simbol Rp dan tanda titik (“Rp 10.000” dan “Rp 15.000”). Kita dapat menggunakan validasi tipe data (type casting) untuk menangani hal ini.Metode teknisnya adalah menghapus teks "Rp " dan tanda titik . terlebih dahulu, lalu mengubah tipenya dari teks (string) menjadi angka (integer) sehingga menjadi 10000 dan 15000 agar bisa dijumlahkan.



	A	B	C	D	E	F	G
1	**Tanggal**	**Produk**	**Harga**				
2	30/12/2026	Penggaris	Rp 10.000	10000			
3	30/12/2026	Buku Tulis	8000	8000			
4	31/12/2026	Pensil 2B	3000	3000			
5	31/12/2026	Buku Gambar	Rp 15.000	15000			
6	31/12/2026	Pensil HB	3000	3000			
7	31/12/2026	Pensil	3000	3000			

Data dengan format yang benar berada pada kolom D dengan asal data dari kolom C.

Rumus yang digunakan adalah:

```
=VALUE(SUBSTITUTE(SUBSTITUTE(C2,"Rp ",""),".",""))
```

Atau menggunakan tanda titik koma (;)

```
=VALUE(SUBSTITUTE(SUBSTITUTE(C2;"Rp ";"");".";""))
```

Penjelasan:

- SUBSTITUTE(C2;"Rp ";"") digunakan menghapus teks "Rp " pada cell C2 (nomor cell yang ingin diubah mohon dapat disesuaikan sesuai dengan kebutuhan).
- Penggunaan Substitute yang kedua yaitu SUBSTITUTE(...;".";"") berfungsi untuk menghapus pemisah ribuan (.).
- VALUE(...) berguna untuk mengubah hasil dari teks menjadi angka (10000).

Data ganda (duplikasi/duplicate)

Data kembar dapat terjadi umumnya apabila Pengguna tidak sengaja mendaftarkan/menyimpan data tersebut dua kali.

Teknik Penyaringan Keunikan (Deduplication) dapat digunakan untuk memecahkan masalah data ganda ini dengan mencari baris yang memiliki nilai kunci (seperti “ID” atau “Email”) yang sama, lalu hanya menyimpan satu data yang paling valid.

Contoh:

	A	B
1	**Nama**	**Email**
2	Anna	anna2010@gmail.com
3	Roni	roni_sidhar@gmail.com
4	Silvia	silvisilvia@gmail.com
5	Silvia	silvisilvia@gmail.com
6	Zaky	muh_zaky@gmail.com

Pelanggan bernama "Silvia" terdaftar dua kali dengan "Email” yang sama. Kita dapat melakukan menghapus data kedua dan menyisakan/tidak menghapus data pertama (keep first).

Apabila menggunakan Excel, dapat dilakukan langkah-langkah sebagai berikut:

- Blok seluruh tabel.
- Buka menu Data.
- Pilih Remove Duplicates.

Atau apabila menggunakan Google Sheets, dapat mengikut petunjuk berikut:

- Data cleanup.
- Pilih Remove duplicates.

Tips: Centang kolom penanda unik seperti "Email" untuk memastikan keakuratan.

Apabila data telah dihilangkan duplikasinya, hasil yang didapatkan akan sebagai berikut (data Silvia hanya ada satu saja):

	A	B
1	**Nama**	**Email**
2	Anna	anna2010@gmail.com
3	Roni	roni_sidhar@gmail.com
4	Silvia	silvisilvia@gmail.com
5	Zaky	muh_zaky@gmail.com

Integrasi Data (Data Integration)

Pembersihan data umumnya perlu dilakukan apabila data digabungkan dan berasal dari beberapa sistem sehingga formatnya tidak konsisten.

Berikut merupakan contoh data yang diambil dari dua buah sistem:

	A	B	C
1	**Nama**	**TglLahir**	**Email**
2	Anna	5/31/2010	anna2010@gmail.com
3	Bene	20-06-2012	BENIRACHMAN@GMAIL.COM
4	Roni	8/16/2013	roni_sidhar@gmail.com
5	Silvia	27-06-2013	silviasilvia@gmail.com
6	Tanudjaja	18-03-2010	TANUDJAJA_T@GMAIL.COM
7	Zaky	3/24/2011	muh_zaky@gmail.com

Data pada baris 2, 4, 5, dan 7 merupakan diambil dari sistem pertama. Data pada baris 3 dan 6 diambil dari sistem kedua. Dapat dilihat bahwa format Tanggal Lahir (“TglLahir”) dan “Email” dari kedua sistem ini tidak konsisten.

Apabila Tanggal Lahir ingin disamakan formatnya dengan menggunakan dd-mm-yyyy, maka solusi pada bagian 2b dapat dilakukan. Mohon agar dapat melihat bagian ini untuk penjelasan lebih detail.

Jika alamat “Email” ingin disamakan penulisannya menjadi huruf kecil semua dan menghilangkan spasi, maka formula =LOWER(TRIM(C2)) dapat digunakan.

Penjelasan:

- TRIM(C2) digunakan untuk menghapus spasi yang berlebihan di awal, akhir, dan spasi ganda di antara kata.
- LOWER(...) berfungsi mengubah seluruh huruf menjadi huruf kecil.

Hasil berikut ini merupakan data yang telah dibersihkan:

	A	B	C	D	E
1	**Nama**	**TglLahir**		**Email**	
2	Anna	5/31/2010	31/05/2010	anna2010@gmail.com	anna2010@gmail.com
3	Bene	20-06-2012	20/06/2012	BENIRACHMAN@GMAIL.COM	benirachman@gmail.com
4	Roni	8/16/2013	16/08/2013	roni_sidhar@gmail.com	roni_sidhar@gmail.com
5	Silvia	27-06-2013	27/06/2013	silviasilvia@gmail.com	silviasilvia@gmail.com
6	Tanudjaja	18-03-2010	18/03/2010	TANUDJAJA_T@GMAIL.COM	tanudjaja_t@gmail.com
7	Zaky	3/24/2011	24/03/2011	muh_zaky@gmail.com	muh_zaky@gmail.com

Masalah Teknis dan Sistem

Berikut adalah beberapa contoh terkait masalah teknis dan sistem:

- Data tidak lengkap atau hilang: dapat terjadi apabila pada sistem terdapat masalah teknis. Penanganan yang dapat dilakukan untuk masalah ini serupa dengan penamangan untuk data yang dikosongkan
- Perbedaan format data seperti tanggal, nomor telepon, mata uang, dan lainnya dapat terjadi, terutama ketika migrasi sistem dilakukan.
- Masalah data ganda dapat terjadi juga apabila sistem eror atau pada sistem tidak terdapat validasi/pengecekan bahwa tidak boleh memasukkan data yang sama dua kali.

Daftar Pustaka

- Noprianto. (2026). *Mengenal dan Menggunakan Bahasa Pemrograman Singkong*. PT. Stabil Standar Sinergi.

Buku ini berisi sejumlah contoh dan penjelasan langkah demi langkah yang mudah dipahami untuk analisis data sederhana, dengan bahasa pemrograman Singkong.

Pembahasan diawali dengan contoh data penjualan yang tidak bersih, yang merepresentasikan kondisi yang umum terjadi, sebagai hasil dari sistem yang berjalan. Sejumlah contoh pembersihan data kemudian dibahas, sebelum pada akhirnya dilanjutkan pada analisis datanya.

Sebagai pelengkap, buku ini juga menyajikan dashboard GUI dengan chart, program entri data sederhana, contoh bekerja dengan sumber data lain (sistem database relasional dan web service), dan contoh laporan sederhana.

Dr. Noprianto mengembangkan bahasa pemrograman Singkong sejak akhir 2019. Beliau menyukai pemrograman, dan mendirikan serta mengelola perusahaan pengembangan software dan teknologi Singkong.dev (PT. Stabil Standar Sinergi). Noprianto menyelesaikan pendidikan doktor dan telah menulis beberapa buku pemrograman (termasuk Python, Java, dan Singkong). Informasi lebih lanjut: noprianto.com

Dr. Buyung Sofiarto Munir saat ini adalah Vice President di PT. PLN (Persero), setelah mengemban berbagai tanggung jawab manajerial di PT. PLN (Persero) selama lebih dari 23 tahun. Selain sebagai seorang praktisi berpengalaman, beliau juga adalah dosen dan peneliti dengan lebih dari 100 publikasi ilmiah di berbagai konferensi dan jurnal internasional. Dr. Buyung memiliki gelar Doktor di bidang ilmu komputer dari Universitas Bina Nusantara, Master of Science dari Delft University of Technology, dan Sarjana di bidang Teknik Elektro dari Institut Teknologi Bandung.

Dr. Maria Seraphina Astriani meraih gelar Doktor dibidang computer science serta memiliki pengalaman profesional dan membantu berbagai perusahaan di Indonesia maupun manca negara khususnya di bidang solusi Information Technology (IT) dengan menggunakan teknologi website/desktop/mobile. Minat penelitiannya meliputi human-computer interaction, IT solution, machine learning, dan pattern recognition. Beliau juga membagikan pengalaman di dalam dunia IT dengan mengajar sejak tahun 2010.
