Data Preparation and Exploratory Analysis

From Raw Files to Trustworthy Tables: Cleaning, Verification, and First Exploration

Dr. Jose Mendoza, Academic Director and Clinical Associate Professor
Version 1.0 · July 2026

Chapter Information

ABSTRACT

This chapter develops data preparation and exploratory analysis as connected disciplines. It presents preparation as a four-stage pipeline — raw, inspected, cleaned, documented — and introduces the chapter's central instrument, the verification log, in which the analyst predicts each cleaning step's effect on row counts and totals before running the step and records the actual values afterward. The chapter treats the major defect families in turn: missing values and the bias each remedy introduces, legitimate extremes against genuine errors, exact and near-duplicate records, and inconsistencies of type, format, and category. It covers the constructive side of preparation — derived variables and deliberate grain changes — and introduces exploratory data analysis as disciplined questioning. A section on AI assistance names the characteristic failure modes of delegated cleaning code. The labs clean StyleCraft's transactions_raw file against its documented defect catalog and certify the result against the transactions_clean answer key.

KEYWORDS

data preparation; data cleaning; verification log; missing data; imputation; outliers; duplicate records; type coercion; exploratory data analysis; marketing analytics

VERSION AND DATE

Version 1.0 · July 2026 · Language: English (United States)

SUGGESTED CITATION

Mendoza, J. (2026). Data preparation and exploratory analysis. In *Applied business analytics for marketing decision-making: Business analytics and data visualization* (Chapter 4, Version 1.0) [Open educational resource]. CC BY 4.0.

COMPANION REPOSITORY
Datasets, notebooks, and figure sources for this chapter: https://github.com/jrmst102/businessanalytics

Chapter Learning Objectives

By the end of this chapter, students should be able to:

1. Describe the preparation pipeline — raw, inspected, cleaned, documented — and explain why cleaning decisions are analytic decisions rather than mechanical chores.
2. Direct an AI assistant to load, inspect, and transform a dataset in Google Colab, and verify the results against predictions recorded in a verification log.
3. Run and interpret the standard inspection routines — shape, data types, head, info, and describe — as the first contact with any new file.
4. Distinguish the three missing-data mechanisms at an intuitive level, choose among dropping, imputing, and flagging, and state the bias each remedy introduces.
5. Identify and handle outliers using interquartile-range fences, z-scores, and domain limits, and defend the distinction between a legitimate extreme and a genuine error.
6. Detect and remove exact and near-duplicate records, and explain why near-duplicates evade the obvious check.
7. Coerce types and standardize formats and category labels, including dates, currency strings, and inconsistent categorical values.
8. Create derived variables and grouped summaries, including deliberate changes of grain, and verify that an aggregation preserves the totals it should preserve.
9. Produce and interpret exploratory visualizations — histograms, box plots, bar charts, and scatterplots — reading them for anomalies, distribution shape, and questions worth pursuing.
10. Recognize the characteristic failure modes of AI-generated cleaning code and audit such code before trusting any row it touched.

Chapter 3 closed with a warning. The data dictionary describes the data as it should be; the files as they arrive are another matter. Chapter 3 supplied the vocabulary for that gap — the six data quality dimensions of Section 3.7 and the defect examples of Table 3.4 — and promised that the next chapter would do the repair. This chapter keeps the promise. It develops the discipline of data preparation: finding the defects a raw file contains, remedying them defensibly, documenting every choice, and verifying the result before a single downstream number is computed from it. It then turns to the first analytical use of a prepared table, exploratory data analysis, in which the analyst interrogates distributions and charts for surprises before committing to any formal method.

CONCEPT

What This Chapter Is Really About

Chapter 3 argued that a number is a claim. This chapter makes the uncomfortable companion argument: before any claim can be computed, someone edits the evidence. Every cleaning decision — this row is a duplicate, that value is an error, these labels are the same category, those records are out of scope — changes the numbers every later chapter will report, and most of these decisions are invisible in the final analysis. An undisciplined analyst makes them silently and forgets them; a disciplined analyst makes them explicitly, predicts their effect before executing them, records predicted against actual in a verification log, and documents every exclusion. The technical content of this chapter is pandas mechanics; the professional content is editorial accountability. In other words, cleaning is not what happens before the analysis. Cleaning is the first analysis.

4.1 Marketing Decision Context: The File That Does Not Add Up

Chapter 3 ended with a small institutional victory: StyleCraft adopted a headline definition of repeat purchase rate, assigned it an owner, and published a data dictionary. The dictionary now records what every column in the company's data family is supposed to mean. Ten days before the quarterly expansion review, the analytics team learns what the columns actually contain.

The setting is the review itself. The Chapter 2 memo committed StyleCraft to revisiting the paused store openings this quarter, with two named analyses in hand, and both analyses — the same-age cohort comparison and the store-economics summary — must be computed from transaction data. To support them, the data engineering contractor has assembled a consolidated extract, transactions_raw, merging order lines from the e-commerce platform, the point-of-sale system in the stores, and the mobile app into a single file at the order-line grain described in Table 3.1. The extract covers the full twenty-four-month history. It is the file from which every number in the review will be computed.

The trouble begins with a routine reconciliation. Before building anything, the analytics lead compares the extract's total revenue for the last complete quarter against the figure the finance system reported for the same quarter — a known total, produced by an audited system, for exactly the period the extract should match. The two figures disagree. The extract is higher, by a margin of a few percentage points: small enough that a hurried analyst might shrug it off, and large enough, on a quarterly revenue base, to change the story the expansion numbers tell. A quick inspection makes matters worse rather than better. Some rows show a quantity of 999. Some prices sit far outside the catalog's $24-to-$90 band, and others arrive not as numbers at all but as text with a currency symbol. The channel column contains “Online,” “online,” and “web,” which the group-by from Chapter 3's lab would happily count as three channels. A spot check against the customers table turns up order dates earlier than the ordering customer's signup date. Furthermore, several blocks of rows look suspiciously like other blocks of rows.

The CFO's question from Chapter 3 — “Which of these numbers is wrong?” — returns in a sharper form. This time the question is not which definition to prefer, but whether the underlying

records can be trusted at all, and the deadline is fixed: the review convenes in ten days, and every analysis on its agenda inherits whatever this file contains. The decision the analytics lead must make is therefore not any single cleaning choice. It is whether, and on what documented basis, to certify a cleaned version of transactions_raw as the table of record for the expansion review — knowing that the CFO will ask not only what the numbers are, but what was removed, what was changed, and how the team knows the cleaned file reconciles to finance's known total.

This chapter is the ten days. Sections 4.2 and 4.3 establish the pipeline and the first inspection. Sections 4.4 through 4.7 treat the defect families one at a time — missing values, outliers, duplicates, and inconsistent types and labels — because each family demands a different diagnosis and carries a different risk. Sections 4.8 and 4.9 turn from repairing the file to using it: deriving variables, changing grain deliberately, and exploring the cleaned data for the surprises that formal analysis should know about in advance. Section 4.10 examines how AI assistants help and fail at exactly this work, and the labs in Section 4.11 perform the certification on StyleCraft's real files, verification log and all.

4.1.1 Opening Case Questions

Keep these questions in mind while reading, and return to them after completing the labs.

- The extract's revenue exceeds finance's known total. Which of the defects listed above could inflate revenue, which could deflate it, and which could do either?
- The analytics lead reconciled against a known total before doing anything else. What other known quantities could serve the same anchoring role for this file?
- If the team simply deleted every suspicious row, the file would reconcile eventually. Why is that not the same as certifying the file?
- Who, besides the analytics team, needs to be able to reconstruct what was removed and why — and what document should make that possible?

4.2 The Preparation Pipeline: Raw, Inspected, Cleaned, Documented

The opening case presents a file that cannot be trusted and a deadline that does not care. The temptation in such moments is to start fixing things — delete the 999s, dedupe, move on. This section argues for a slower first move: adopting a pipeline that separates the stages of preparation and preserves the evidence at every stage.

Data preparation, as this guide uses the term, is the disciplined transformation of raw source data into a documented, analysis-ready table, through the repeatable sequence raw → inspected → cleaned → documented. Each stage has a distinct obligation. The raw stage preserves the file exactly as it arrived; the raw file is never edited in place, because it is the evidence against which every later claim of "we fixed X" is checked. The inspected stage produces a written account of what the file contains — its shape, types, ranges, and apparent defects — before anything is changed, so that remedies respond to diagnoses rather than to habit. The cleaned stage applies remedies one defect family at a time, in code, in an order that is recorded and can be re-run from the untouched raw file. The documented stage records what was done and why: which rows were

removed and under what rule, which values were changed or filled, which decisions were judgment calls, and how the result was verified. The practitioner literature converges on the same broad sequence and on a durable practical observation: preparation often consumes a larger share of analytical work than modeling itself (Dasu & Johnson, 2003; Anaconda, 2020).

DEFINITION

Data Preparation

Data preparation is the disciplined transformation of raw source data into an analysis-ready table through a repeatable, documented sequence of inspection, cleaning, and verification steps, such that the resulting table can be reproduced from the untouched raw file and every change to the data can be traced to a recorded decision.

Source: Adapted from Dasu and Johnson (2003) and Hellerstein (2008).

Two properties of the pipeline deserve emphasis, because they distinguish professional preparation from improvised tidying. The first is reproducibility. Because every remedy is a step in code applied to the preserved raw file, the entire cleaned table can be regenerated at will — after a bug is found, after a rule is revised, after a stakeholder asks "what happens if we keep those rows?" An analyst who cleaned by hand-editing a spreadsheet can answer none of these. In code, reproducibility has a concrete first line: the raw object is loaded once and immediately copied, and every transformation is applied to the copy, so that the baseline against which the log's predictions are checked is still in memory at the end of the pipeline. The labs in Section 4.11 make that copy the first executable step, and Exercise 4.7 shows what goes wrong when it is skipped.

The second property is order-awareness. Cleaning steps interact: deduplicating before fixing the currency-string prices means the duplicate check compared text to text; filling missing values before removing duplicates means some fills were computed from rows about to be deleted. There is no single correct order for every file, but there is always a chosen order, and the choice belongs in the documentation. As a default, this guide inspects first, fixes types and labels second (so that later comparisons compare like with like), removes duplicates third, resolves impossible values and outliers fourth, treats missing values fifth, and derives new variables last, verifying after every step.

4.2.1 The Verification Log

The previous paragraphs used the word "verify" repeatedly without saying how. This subsection introduces the chapter's central instrument, which is also this chapter's expression of the predict-then-verify discipline from Section 1.7: the verification log.

A verification log is a running record, kept alongside the cleaning code, in which the analyst writes down — before executing each step — what the step is expected to do to the file's checkable quantities, and then records what actually happened. The checkable quantities are deliberately humble: the row count, the count of distinct orders and customers, the sum of

revenue, the count of missing values in the affected columns. If the plan is to remove exact duplicate order lines, the log first records the number of exact duplicates found during inspection and the predicted row count after removal; the step is then run, the actual count is recorded, and any gap between prediction and outcome must be explained before the pipeline proceeds. A mismatch is not an annoyance; it is the log working. Every mismatch means the analyst's model of the data or the analyst's model of the code is wrong, and both are exactly the things worth knowing before the expansion review, not after.

The log earns its keep twice more downstream. At certification time, it is the answer to the CFO's question: the sequence of entries, read top to bottom, is a complete account of how the raw file became the table of record, including the reconciliation against finance's known total. Certification is a threshold, not a mood — a file is certified for a given use when every step's prediction has been confirmed or explained and no quantity the use depends on remains unresolved, and Section 4.11 writes that threshold as a line of code rather than a sentence of reassurance. And whenever an AI assistant writes the cleaning code — the normal case in this course — the log is the audit that makes delegation safe, a point Section 4.10 develops.

CONCEPT

The Verification Log

For every cleaning step, record four things before running it: the step's rule in one sentence; the quantities it should change (rows, totals, missing counts) and by how much; the quantities it should leave untouched. Run the step; record the actual values; explain any difference before continuing. A cleaned file is certified for a stated use when the log runs from the raw file's baseline to the final table with every step's prediction either confirmed or explained, and no quantity that use depends on is left unresolved.

Source: Course concept developed for this guide, informed by Dasu and Johnson (2003) and Hellerstein (2008).

Table 4.1 summarizes the pipeline and shows where the log sits in it. With the frame established, the next section performs the pipeline's first stage on a new file: inspection.

Table 4.1

The preparation pipeline and its obligations

Stage	Obligation	Key artifact	Cardinal sin
Raw	Preserve the file exactly as it arrived	The untouched raw file	Editing the raw file in place
Inspected	Diagnose before treating: shape, types, ranges, defects	Written inspection notes; verification-log baseline	Fixing on sight without recording the diagnosis

Stage	Obligation	Key artifact	Cardinal sin
Cleaned	Remedy one defect family at a time, in code, in a recorded order	The cleaning notebook; one log entry per step	Silent steps whose effects were never predicted
Documented	Make every change traceable and the result reproducible	The completed verification log; dictionary updates	A cleaned file whose differences from raw cannot be explained

4.3 First Contact: Inspection Routines

The pipeline's first active stage is inspection, and this section makes it concrete. The goal of inspection is a written diagnosis: what the file contains, how it is typed, what ranges its values occupy, and which of the defect families from Sections 4.4 through 4.7 appear to be present. Nothing is changed at this stage. In Chapter 3's editorial metaphor, this is the read-through before any correction is marked.

Five routines, all one-liners in pandas, do most of the work, and they should be run in roughly this order on any new file. The shape attribute reports rows and columns; the row count becomes the first line of the verification log, the baseline every later count is checked against. The dtypes attribute reports the stored type of every column, to be compared — predict first, then look — against the types the data dictionary says each column should have; Chapter 3's lab showed how much measurement information the stored types can silently discard, and a date arriving as text or a price arriving as object is often the first visible symptom of the format problems Section 4.7 treats. The head method displays the first rows, which is where a human eye catches what summaries miss: a currency symbol in a numeric column, a label in the wrong case, a column that is obviously two facts packed into one. The info method combines shape, types, and — critically — the count of non-null values per column, turning missing data from a suspicion into a per-column tally. Finally, the describe method reports count, mean, standard deviation, minimum, quartiles, and maximum for numeric columns, and its extremes are the fastest defect detector available: a maximum quantity of 999 or a minimum price below the catalog floor announces itself in one line of output.

Two habits convert these routines from ritual into diagnosis. The first is prediction. Before running describe on unit_price, the analyst who has read the data dictionary knows the catalog band is $24 to $90 and writes that expectation down; the output is then read as a comparison, not a revelation. This is predict-then-verify (Section 1.7) applied at the cheapest possible point, before any code that changes anything has been written. The second is anchoring against known totals. Somewhere outside the file there is usually a number the file must agree with — finance's quarterly revenue, the loyalty platform's member count, last month's certified report. Computing the file's version of that number during inspection, as StyleCraft's analytics lead did, establishes at the outset how far from reconciled the raw file is, and gives the final log entry its target.

Inspection ends with a short written list: the defects observed, the columns affected, and the apparent scale of each problem. For StyleCraft's extract, that list will name missing values, outliers, duplicates, and inconsistent types and labels — the four families the next four sections treat in turn, beginning with the defect that is hardest to see because it is not there: the missing value.

4.4 Missing Values: Mechanisms, Remedies, and the Bias Each Introduces

Inspection produces, among other things, a per-column count of missing values. This section asks the two questions that count cannot answer: why are the values missing, and what should be done about them? The order of the questions matters. The remedy that is defensible depends on the mechanism that produced the absence, and analysts who reach for a remedy first are treating a symptom whose cause they have not diagnosed.

Begin with what Chapter 3 already established. Section 3.7 noted that in StyleCraft's transactions a missing campaign_id can mean two opposite things: the order was legitimately unattributable to any campaign (an informative null — the absence is the value), or the attribution pipeline failed (a true gap — the value existed and was lost). The two absences are indistinguishable in the file and opposite in meaning, and no amount of pandas can tell them apart; only knowledge of the source system can. The first move with any missing values is therefore not technical at all: consult the data dictionary (Section 3.8), whose known-issues field exists to record exactly this, and the system owners, who hold what no file contains.

For absences that are genuine gaps, the statistical literature distinguishes three mechanisms, and this guide needs them only at the level of intuition (Rubin, 1976; Little & Rubin, 2019). A value is missing completely at random (MCAR) when the fact that it is missing has nothing to do with anything — as if the recording system dropped values by coin flip. A value is missing at random (MAR) when the missingness is related to other observed columns but not to the missing value itself: if in-store customers skip the optional age question more often than online customers do, age_band is MAR given channel. A value is missing not at random (MNAR) when the missingness depends on the value that would have been recorded: if customers with the highest incomes are the most likely to decline an income question, the absences themselves are systematically hiding one end of the distribution. The names are awkward — MAR does not mean “random” in the everyday sense — but the escalation matters. Under MCAR, the observed rows are a smaller but unbiased sample. Under MAR, the observed rows are biased, and the bias can in principle be corrected using the observed columns. Under MNAR, the observed rows are biased in a way the data cannot itself reveal or repair.

DEFINITION

Missing-Data Mechanisms

The missing-data mechanism is the process by which absences arise in a dataset. Values are missing completely at random (MCAR) when missingness is unrelated to any variable; missing at random (MAR) when missingness is related to observed variables but not to the missing value itself; and missing not at random (MNAR) when missingness depends on the unobserved value. The mechanism, not the count of absences, determines which remedies are defensible and what bias they carry.

Source: Adapted from Rubin (1976) and Little and Rubin (2019).

Three families of remedy are available, and each purchases convenience with a specific bias that the analyst must be able to name. Dropping removes the affected rows (or, rarely, the affected column). It is simple, honest, and, under MCAR, costs only sample size. Under MAR or MNAR, however, dropping deletes a nonrandom slice of the world: if in-store shoppers disproportionately lack age_band, dropping those rows quietly removes in-store shoppers from every later analysis that uses age — precisely the customers the suburban expansion question is about. Imputing fills the gaps with estimated values — the column's mean or median, the most frequent category, or a value predicted from other columns. Imputation preserves rows, but it manufactures data: filling with the mean shrinks the column's variability and overstates confidence in later estimates, and filling categorical gaps with the most frequent label inflates the majority category (Little & Rubin, 2019; Osborne, 2013).

Flagging adds a companion indicator column — age_band_missing — so that the absence itself becomes analyzable, and it is normally combined with one of the other two rather than used alone. Flagging is the least glamorous remedy and frequently the most useful, because it preserves the one fact dropping and imputing both destroy: which rows were touched. It is not, however, free, and it is not a remedy in its own right. A flag does not resolve the missingness; it records it. It adds a column that every downstream reader must understand, and flagging every column indiscriminately produces a table full of indicators nobody interprets — and, in the predictive chapters ahead, features that can leak information about the outcome if the missingness is itself a consequence of it. The discipline is to flag where the absence is analytically interesting or where a judgment call needs to be testable later, and to say so in the log. Table 4.2 summarizes the trade.

Table 4.2

Missing-value remedies and the bias each introduces

Remedy	What it does	Defensible when	Bias it introduces
Drop rows	Removes rows with missing values in scoped columns	Mechanism plausibly MCAR; affected share small; rows not central to the question	Deletes a nonrandom slice under MAR/MNAR; shrinks sample
Impute (mean/median)	Fills numeric gaps with a central value	Numeric column; small gap share; downstream use tolerates damped variability	Understates spread; distorts distribution shape; fabricates precision
Impute (most frequent)	Fills categorical gaps with the modal label	Categorical column; gap share very small	Inflates the majority category; can flip group comparisons
Impute (model-based)	Predicts the missing value from other columns	MAR plausible; the imputation model is itself verified	Imports the model's assumptions; harder to audit
Flag	Adds an indicator column marking absence	Often useful for auditability, usually alongside a justified remedy	Does not alter observed values, but does not repair the missingness; the indicator must be interpreted and governed, and indiscriminate flagging can leak information into later models

Two closing disciplines connect this section back to the pipeline. First, every missing-value decision is a log entry: the count of absences before, the rule applied, the count after, and — for drops — the number of rows removed, predicted and then confirmed. Second, every decision is a documented exclusion or alteration in the sense Section 4.14 will examine ethically: filling in-store shoppers' ages with the online majority's median is an editorial act with consequences for whose behavior the analysis represents, and it belongs in writing. However the gaps are treated, the next defect family is easier to see and harder to explain away: values that are present but implausible.

4.5 Outliers: Legitimate Extremes and Genuine Errors

Missing values are absences; outliers are presences that strain belief. This section develops the chapter's second defect family, and it begins with the distinction on which every outlier decision turns: an outlier is a value that is far from the rest, and being far from the rest is not, by itself, evidence of being wrong.

An outlier is an observation that lies unusually far from the bulk of the data (Barnett & Lewis, 1994). The definition is deliberately about distance, not error, because extreme values arise in two ways that demand opposite treatment. Some extremes are legitimate: StyleCraft's occasionwear baskets are designed to be large, and a suburban customer buying four dresses at

the top of the $24-to-$90 band produces a big — and real — order line. Deleting such rows does not clean the data; it amputates the phenomenon, and in StyleCraft's case it would amputate the exact behavior the expansion review needs to see. Other extremes are errors: a quantity of 999 in an apparel transaction is not a large purchase, it is a sentinel or entry error, and a unit price of $2,400 in a catalog that tops out near $90 is a decimal slip (Table 3.4 previewed both). The analyst's job is not to remove outliers; it is to classify extremes as legitimate or erroneous, using evidence, and then treat only the errors.

Detection comes first, and three rules cover most practice. The interquartile-range rule, inherited from Tukey's exploratory tradition, computes the first and third quartiles and flags values below $Q1 - 1.5 \times IQR$ or above $Q3 + 1.5 \times IQR$ (Tukey, 1977); it is robust, distribution-light, and it is the rule the box plot of Section 4.9 renders as a picture. The z-score rule flags values more than some threshold — conventionally three — standard deviations from the mean; it is familiar but fragile, because the outliers being hunted inflate the very mean and standard deviation used to hunt them, and it presumes a roughly symmetric distribution that marketing data rarely provides. Domain limits flag values outside ranges the business itself declares: prices outside the catalog band, discounts above the maximum the promotion engine permits, quantities beyond any plausible basket. Domain limits are the most powerful rule of the three, because they encode knowledge rather than statistics — and they come straight from the data dictionary's allowed-values field (Section 3.8), which is one more argument for writing that field down.

Classification and treatment follow detection. For each flagged value, the analyst seeks corroboration: does the extreme row make sense as a whole (a large line_revenue supported by a plausible quantity and an in-band price is a basket; a large line_revenue produced by quantity 999 is not)? Does the source system have a known sentinel convention? Does the same customer or store generate other extremes? Errors are then treated — corrected where the true value is recoverable (the $2,400 that is demonstrably $24.00), set to missing where it is not (converting the problem to Section 4.4's, with the mechanism known), or excluded with a documented rule — while legitimate extremes are kept, and kept visibly: the verification log records how many values each rule flagged, how each flagged value was classified, and what the treatment changed. Winsorizing — capping extremes at a percentile rather than removing them — appears in practice, but this guide treats it with caution: it alters real values wholesale, and in a dataset whose most decision-relevant behavior lives in the right tail, capping the tail is editing the story.

DEFINITION

Outlier

An outlier is an observation that lies unusually far from the bulk of the data, as judged by a stated rule such as the interquartile-range fences, a z-score threshold, or domain limits. An outlier is a candidate for investigation, not for removal: the analyst must classify it as a legitimate extreme, to be retained, or a genuine error, to be corrected, set to missing, or excluded under a documented rule.

Source: Adapted from Barnett and Lewis (1994) and Tukey (1977).

The discipline, restated: detect with a rule, classify with evidence, treat only errors, log everything. The next defect family requires no classification judgment at all — a duplicated order line is simply wrong — but it compensates by being harder to find than it looks.

4.6 Duplicate Records: Exact and Near

Outliers at least announce themselves in a describe output. Duplicates hide in plain sight: every individual value in a duplicated row is perfectly plausible, and the defect exists only in the relationship between rows. This section treats the two forms duplication takes and the reason the second form evades the first form's check.

An exact duplicate is a row identical to another row in every column — in a transaction file, the same order line recorded twice, typically through a double export, a retried upload, or a merge that ingested the same source twice. Exact duplicates are mechanically easy: pandas' duplicated method flags them, drop_duplicates removes all but the first occurrence, and the verification log entry is arithmetic — if inspection found forty-one duplicated rows, the cleaned file must have exactly forty-one fewer, and the removed revenue must equal the sum of the removed lines. The analytical stakes, however, are anything but minor, because duplicated order lines inflate revenue, order counts, and every per-customer metric downstream — and inflation is precisely the direction of StyleCraft's reconciliation gap in Section 4.1. Uniqueness, recall, is one of Section 3.7's six quality dimensions; this section is its repair shop.

A near-duplicate is the subtler form: two rows that record the same real-world event without being identical in every column. StyleCraft's extract contains the classic case, previewed in Table 3.4 — a block of orders re-exported from a source system with a new export timestamp, so that every substantive column matches but one administrative column differs. A check for fully identical rows sails past these untouched, which is exactly why they were planted. Detecting near-duplicates requires the analyst to decide which columns define the identity of an event: if two rows share order_id, order_line_id, customer_id, product_id, quantity, and price, they describe the same order line no matter what a load timestamp says, and duplicated with a subset argument restricted to the identity columns finds them. That decision — which columns constitute identity — is a judgment about the business, not about pandas, and it is the reason near-duplicate removal must be documented rather than merely executed: a subset chosen too narrowly deletes real repeat purchases (the same customer genuinely buying the same product twice in one day is rare but real), while a subset chosen too broadly leaves the inflation in place.

Duplication has a second and less obvious source, and it is worth naming here because it is the failure the labs guard against explicitly: duplicates the analyst creates. A join to a dimension table that is supposed to hold one row per customer will silently multiply transaction rows if the dimension in fact holds two, and the multiplication inflates revenue without raising an error or leaving an unmatched row to count. The remedy belongs to the same discipline as the rest of this section: state the expected cardinality of every join before running it, have the code enforce it, and check the row count on both sides. Section 4.11 shows the pattern, and Section 4.10 explains why AI-drafted merge code makes the check non-negotiable.

The verification log carries the burden of proof in every case. For each deduplication step: the number of rows flagged, the identity rule in one sentence, the predicted row count after removal, the actual count, and the change in total revenue — which, at the end of the pipeline, should account for a large share of the reconciliation gap the opening case discovered. If it does not, the log has just generated the chapter's most useful kind of finding: the gap has another cause, and the analyst knows to keep looking. With rows now unique, the pipeline turns to defects inside the values themselves: types, formats, and labels that disagree about how to spell the same fact.

4.7 Types, Formats, and Inconsistent Categories

Deduplication compares values, and comparison presumes the values are comparable. This section treats the defect family that breaks that presumption: values stored in the wrong type, formats that vary across source systems, and category labels that spell one fact several ways. Chapter 3 drew the conceptual line — stored data types are not measurement levels, and files routinely discard measurement information (Section 3.4.1) — and this section does the repairs that follow from it.

Type coercion is the conversion of a column from its stored type to the type its measurement requires: text to number, text to date, free text to categorical. The need arises because consolidated extracts inherit the habits of their sources, and operational systems export whatever their own storage held. StyleCraft's extract exhibits the canonical case: prices arriving as text with stray currency symbols — “$45.00” where 45.00 was meant — which a naive sum would refuse to add or, worse in other tools, silently drop. The repair pattern is normalize, then strip, then convert, then verify: put the column into a known text type first, because a consolidated extract frequently contains a mixture of strings and already-numeric values and a bare string accessor fails or misbehaves on the mixture; remove the non-numeric characters; convert with pd.to_numeric; and inspect what refused to convert. The final clause is where discipline lives. Conversion functions offer an errors=“coerce” convenience that turns unconvertible values into missing values, and used blindly it is a silent-data-loss machine: every stubborn string becomes a NaN without a word of complaint (pandas development team, 2026d). The verified pattern predicts how many values should fail conversion (from inspection), coerces, counts the new missing values, and requires the two numbers to match — a verification-log entry like any other.

Coercion has a companion problem that the same discipline solves: a numeric column that must also carry missing values. Setting a sentinel quantity to missing in an ordinary integer column silently converts the whole column to floating point, so that counts of units begin displaying as 1.0 and 2.0 and a column that measures a count stops looking like one. Pandas provides a nullable integer type for exactly this situation, and declaring it before introducing the missing value preserves the measurement meaning Chapter 3 insisted on (pandas development team, 2026b). The point is not cosmetic. A column's type is a claim about what kind of quantity it holds, and a repair that quietly changes the claim is a defect introduced by the cleaning.

Dates deserve the same care doubled, because they fail in more ways: multiple formats across source systems, day/month ambiguity, and values that parse cleanly into impossible facts. Chapter 3's conformity examples — an order_date before the customer's signup_date, or before the store's opening_date — parse without complaint; only a cross-table check against customers and stores exposes them, and the lab performs exactly that check.

Categorical standardization is the repair of labels that disagree about the same fact. StyleCraft's channel column contains “Online,” “online,” and “web”; its category labels include both “Occasionwear” and “occasion wear.” Chapter 3 named this a consistency defect and warned what a naive group-by does with it: three channels reported where one exists, with revenue split among them — a defect that does not change totals but silently corrupts every comparison, share, and chart built on the affected column. The repair is a mapping: an explicit, written dictionary from every observed variant to its canonical label, applied in code, with the log recording the value counts before and after and confirming that the total row count never moved. The mapping's authority is, once again, the data dictionary's allowed-values field: the canonical labels are not the analyst's invention but the documented vocabulary of the column. Two practices keep standardization honest. Normalize case and whitespace first, so the mapping enumerates real variants rather than typographical ones; and never map a variant whose meaning is uncertain — “web” probably means “Online,” but if any chance exists that it marks a distinct legacy channel, that question goes to the source-system owner, not to a guess.

DEFINITION

Type Coercion

Type coercion is the conversion of a column from its stored data type to the type required by its intended measurement level — for example, currency strings to numeric values, or date-like text to true dates — performed with an explicit accounting of every value that fails to convert. Coercion that silently discards or nullifies unconvertible values, or that changes what kind of quantity a column claims to hold, is a data defect of its own.

Source: Adapted from McKinney (2022) and Hellerstein (2008).

With rows unique, values typed, labels canonical, gaps treated, and extremes adjudicated, the file is clean in the sense this chapter can defend: every difference from the raw file is a logged, explained decision. The pipeline's remaining work is constructive rather than corrective, and the next section turns to it — building the new columns and new grains the analyses ahead will need.

4.8 Derived Variables and Deliberate Grain Changes

Cleaning removes and repairs; preparation also builds. This section covers the two constructive operations that end most preparation pipelines: deriving new variables at the existing grain, and aggregating to a coarser grain on purpose. Both are routine pandas; both change what the table can claim; and the second, done carelessly, is the most consequential silent error in analytics.

A derived variable is a new column computed by an explicit rule from existing columns, at the grain of the table in which it appears. StyleCraft's schema already contains two — line_revenue is defined as quantity × unit_price × (1 − discount_pct), and line_cost as quantity × unit_cost — and Chapter 3's dictionary entry for line_revenue gave the standing advice this section repeats: recompute derived values rather than trusting stored ones. In a freshly cleaned file the recomputation is also a verification: computing line_revenue from its now-coerced, now-plausible inputs and comparing against the stored column confirms, row by row, that the type repairs and outlier treatments left the file arithmetically coherent. Beyond recomputation, preparation typically derives the small workhorse columns later chapters lean on: an order month extracted from order_date; a full-price flag from discount_pct equal to zero; a line-margin column from line_revenue minus line_cost. Each derivation is one line of code and one dictionary entry — a derived column without a recorded rule is tomorrow's mystery column — and each is logged with the simplest of checks, that the new column contains no unexplained missing values and the row count is unchanged.

Grouped aggregation changes the grain deliberately: it takes a table at one grain and produces a table at a coarser one, using pandas' groupby to collect rows sharing a key and summarize each group. The mechanics are compact — group transactions by order_id and sum line_revenue, and the result is an order-grain table with one revenue per order; group by customer_id and the result is customer-grain; group by store and month and the result feeds the time series of Chapter 10. The discipline is everything Chapter 3's Section 3.3 promised it would be. A grain change changes the claim: average line_revenue at line grain answers a different question than average order value at order grain, and computing “average order value” on the wrong grain is not a rounding disagreement but a different number with a different meaning. Every groupby therefore begins by naming its output grain — one row will equal one what — and choosing the summary each column deserves at that grain: quantities sum, dates take the first or the last, categorical attributes must be constant within the group or must not be carried along at all.

Aggregation gets its own verification pattern, and it is the most satisfying entry in the log because it is exact: totals that should survive the grain change must survive it. The sum of order-level revenue must equal the sum of line-level revenue to the cent; the count of distinct order_id values in the fact table must equal the row count of the order-grain table; the number of customers in a customer-grain roll-up must equal the number of distinct customers in the lines. When a predicted-equal total comes back unequal, the usual culprits are a key column with missing values — pandas excludes missing group keys by default, so groups vanish without comment unless the behavior is made explicit (pandas development team, 2026a) — or rows excluded upstream without a log entry. Aggregation also has a quieter trap that no total will reveal: a sum over a group whose values are all missing returns zero rather than missing, so an order whose only line has an unresolved quantity arrives at order grain reporting revenue of $0.00. The total still reconciles, and a genuinely unknown quantity has been converted into a confident zero. Section 4.11 shows how to keep the unresolved state visible. Either way, the

mismatch, or its suspicious absence, is the log doing its job: it has caught, before any analysis, a discrepancy that would otherwise surface in a meeting.

DEFINITION

Derived Variable

A derived variable is a column computed from existing columns by an explicit, recorded rule, at the grain of the table in which it appears. Some derived variables are computed directly from columns at the current grain — line_revenue from quantity, unit_price, and discount_pct. Others exist only after an explicit aggregation has produced a new table at a coarser grain, as a customer's order count does in a customer-grain roll-up of transactions. A derived variable is complete only when its rule and its grain are documented in the data dictionary and its computation is verified against the totals it should preserve.

Source: Adapted from Kimball and Ross (2013) and Wickham (2014).

The file is now not only clean but productive: it carries the derived columns and companion grains the coming analyses need, and every one of them is documented. The pipeline's corrective and constructive work is done. What remains — before any formal method is chosen — is to look at what the data actually shows, and the next section takes up that discipline.

4.9 Exploratory Data Analysis: Disciplined Questioning

Preparation ends with a table the analyst can defend; analysis should not begin until the analyst has met it. This section introduces exploratory data analysis (EDA), the practice of systematically interrogating a fresh dataset — its distributions, its groups, its relationships — before committing to any formal method. The stance comes from Tukey, who argued that data analysis had overinvested in confirming hypotheses and underinvested in the prior work of finding out what the data suggests: exploratory work, in his phrase, is “detective work” (Tukey, 1977). The detective framing is exact. Exploration is not aimless looking; it is the disciplined pursuit of questions — What does this variable's distribution look like? Is that what the business story predicts? What is that bump? Who are those points? — in which every chart is produced to answer a stated question and every surprise is either explained or written down as a finding (Behrens, 1997).

EDA belongs in this chapter, adjoining preparation, for two reasons that practice keeps confirming. First, exploration is the final stage of verification: a distribution that looks wrong is often a defect the cleaning missed, and the fastest way to discover that a deduplication rule failed is to see the ghost of it in a histogram. Second, exploration is the first stage of every later chapter: the descriptive comparisons of Chapter 5, the segments of Chapter 6, and the models of Part II all begin from facts about shape and spread that exploration establishes. The habit this guide asks for is modest and permanent: no formal analysis on a table the analyst has not explored.

DEFINITION

Exploratory Data Analysis

Exploratory data analysis (EDA) is the systematic examination of a dataset's distributions, groups, and relationships — through summaries and simple charts — undertaken before formal modeling, in order to discover structure, surface anomalies, and generate the questions that later analysis will answer.

Source: Adapted from Tukey (1977) and Behrens (1997).

4.9.1 Distribution Shape: Skewness and Modality

The first exploratory question for any numeric variable is the shape of its distribution, and two aspects of shape do most of the descriptive work. Skewness is asymmetry: a right-skewed distribution has most of its values at the low end and a long tail of large values stretching right, and a left-skewed distribution mirrors that. Marketing quantities are chronically right-skewed — order values, customer spend, campaign clicks all pile up small and tail off large — and StyleCraft's order revenue is right-skewed by design: most baskets are one or two items in a catalog that tops out near $90, while suburban occasionwear baskets stretch the tail. Recognizing skew matters immediately for verification (a symmetric order-value distribution in this data would itself be an anomaly worth chasing) and matters shortly for description, because summarizing skewed data raises choices Chapter 5 takes up. Modality is the number of humps: a unimodal distribution has one concentration of values, a bimodal distribution two. Bimodality is one of exploration's most valuable alarms, because it usually means the variable is describing a mixture of populations rather than one — and in StyleCraft's data, where two customer populations were designed in, a two-humped spending distribution is not noise but the company's central strategic fact showing itself in a histogram.

4.9.2 Reading Exploratory Charts for Anomalies

Exploration works through four chart types, and this guide asks them to be read, at this stage, in a particular way: for anomalies and questions, not for polish. The theory of chart selection and visual encoding — why these forms work and how to design them for an audience — is Chapter 12's territory; here the charts are instruments, and the skill is knowing what each instrument detects.

A histogram shows one numeric variable's distribution by binning values and drawing a bar per bin. Read it for shape (skew, modality, per Section 4.9.1), for spikes (a bar taller than its neighbors at a suspiciously round value — a price exactly at a sentinel, a quantity exactly at a cap), and for gaps and orphans (empty ranges, or a lone bar far from the body, which is an outlier rule made visible).

A box plot draws the median and the quartile box, and it is Section 4.5's IQR rule made visible (Tukey, 1977). Under the common Tukey convention that plotting libraries implement by

default, the whiskers do not sit at the fences themselves: they extend to the most extreme observed values that still lie within 1.5 interquartile ranges of the first and third quartiles, and observations beyond them are drawn individually as possible outliers (Matplotlib Development Team, 2026). The distinction matters when reading a picture as evidence, because the whisker tip is a real observation in the data rather than a computed boundary. The box plot's particular strength is comparison: box plots of order value by store type put the suburban-urban contrast, spread and outliers included, in one glance. Read it for medians that differ across groups, boxes of very different heights (heterogeneity across groups), and dense swarms of flagged points (a tail, or a defect).

A bar chart compares a summary statistic across categories; at the exploratory stage its most important service is humble — a bar per category label is the fastest way to see that "Online," "online," and "web" survived the cleaning, or that one store's revenue is implausibly zero. Read it for categories that should not exist, categories missing that should exist, and magnitudes wildly out of family. A scatterplot shows the joint behavior of two numeric variables, one point per row. Read it for the expected structure (line_revenue rising with quantity along plausible price bands), for points off the structure (a high revenue at quantity one beyond any catalog price — an arithmetic defect surviving), and for clumps that suggest subpopulations. In every case the reading ends the same way: each anomaly becomes a question, each question gets pursued to an explanation — a defect, a documented business fact, or a genuine discovery — and the discoveries are written down, because an insight that lives only in a notebook scroll is not yet evidence.

Exploration, so practiced, closes the preparation story and opens the analysis story. Before the labs perform both on StyleCraft's files, one more section addresses the collaborator that will write most of the code: the AI assistant.

4.10 AI as a Data Preparation Assistant

The pipeline of Sections 4.2 through 4.9 is, line for line, the kind of work AI assistants perform fluently: loading files, coercing types, mapping labels, deduplicating, grouping, plotting. Consistent with the guide's stance (Sections 1.9, 2.8, and 3.9), this section identifies where that fluency genuinely helps, where delegation must stop, and how this chapter's verification theme — the log — makes the collaboration safe.

The genuine help is broad. An assistant given the data dictionary and a description of a defect writes serviceable cleaning code in seconds: the strip-and-coerce pattern for currency strings, the subset-keyed deduplication, the groupby with the right aggregations. It drafts inspection summaries, proposes standardization mappings from observed value counts, generates the boilerplate of every exploratory chart, and — often underrated — explains unfamiliar code the analyst has inherited. For the mechanical majority of preparation, the assistant is a competent junior pair of hands, and this course expects students to use it as one.

The failure modes are equally specific, and three deserve names because the lab plants encounters with each. Silent row drops: cleaning code frequently discards rows as a side effect

— errors="coerce" followed by a dropna, a filter written wider than intended, a groupby key containing missing values — and AI-generated code is prone to including such steps helpfully and without comment, so that the file shrinks and nothing announces it. Wrong join keys: asked to check order dates against signup dates, an assistant must join transactions to customers, and a join on the wrong key, the wrong key type (a customer_id string in one table, a stray-whitespace variant in the other), or a dimension table that is not in fact unique on the key produces a result that runs, returns rows, and is quietly wrong — the most dangerous kind of wrong, because nothing errors and, in the duplicated-key case, the row count moves in the direction that flatters revenue. Unstated imputation defaults: asked to "handle" missing values, assistants default to a remedy — usually mean or modal filling — without surfacing that a choice among Section 4.4's remedies was made at all, importing the corresponding bias unannounced. The common thread is the one this guide keeps finding: the assistant optimizes for code that runs and output that looks complete, while the analyst is accountable for what the code did to the data — and in preparation, what the code did to the data is the entire question.

The countermeasure is the verification log, applied to delegated code exactly as to handwritten code — which is to say, the log is not extra work occasioned by AI; it is the same discipline, now doing double duty as an audit. Before running any AI-generated cleaning step, the analyst writes the prediction: what the step should change, what it should not, and by how much. After running it, the actuals either match or the code goes back. Row counts before and after every delegated step are non-negotiable; a join is always accompanied by a stated expected cardinality, an enforced check that the code makes the join keep it, and a count of matched and unmatched rows; an imputation is always preceded by the question "which remedy is this, and why?" Every exchange is documented per Appendix D.

AI IN PRACTICE

Delegate the Code, Never the Count

A productive pattern for delegated cleaning: paste the relevant data dictionary entries and a sample of rows (never confidential data; the synthetic StyleCraft files are safe), then prompt — "Write pandas code to [one cleaning step]. Before the code, state exactly what the step will change: which columns, how many rows affected, what the row count and revenue total should be afterward, and what the code deliberately leaves alone. If the step is a join, state the expected cardinality and enforce it in the code. List any rows it would drop and why." Run the code only after recording the assistant's predictions and your own in the verification log; if the assistant cannot state what its code will do to the row count, that is the audit failing before the code runs. Compare actuals, investigate every mismatch, and document the exchange per Appendix D.

One boundary completes the section. The assistant may draft every line of cleaning code in this chapter; it may not make the editorial decisions the code executes. Whether "web" means "Online," whether the 999s are sentinels, whether in-store shoppers' missing ages are dropped, filled, or flagged, which columns define a duplicate — these are judgments about StyleCraft's

business, they are exactly the decisions Section 4.14 will argue carry ethical weight, and they belong to the analyst of record. The labs that follow are structured around that division of labor.

4.11 Hands-On Application in Python and Google Colab

The preceding sections built the pipeline conceptually. This section runs it. Lab 4.1 works through the full preparation sequence twice: first on a fourteen-row miniature of transactions_raw constructed in the notebook, so that every count and total can be verified by hand, and then on StyleCraft's full transactions_raw file from the companion repository, against the same defect families at scale. Lab 4.2 explores the cleaned file. Throughout, follow the course division of labor: AI assistants may draft the code (Appendix C has prompting templates; Appendix A covers Colab mechanics), but every step gets a verification-log entry — prediction first — and every AI exchange is documented per Appendix D.

4.11.1 Lab 4.1, Part A: The Dirty Dozen (and Two More)

The miniature below contains fourteen order lines and, planted among them, one instance of each defect family from the chapter: a currency-string price, two label variants in channel, an exact duplicate, a suspected re-export pair that also contains an error, a quantity sentinel, an out-of-band price, a missing discount, a missing campaign attribution, and an order date that precedes the customer's signup. Before running the cell, read the code and start the verification log: predict the shape, and predict which routine from Section 4.3 will surface which defect.

Note the cell's last two statements before the shape check. The raw frame is built once and then copied, and every step that follows operates on the copy, tx. This is the raw-stage obligation of Table 4.1 written as executable code: at the end of the pipeline, tx_raw still holds the currency strings, the label variants, and the 999, so any log entry can be re-checked against the evidence it claims to have repaired.

Code 4.1. Create the transactions_raw miniature and preserve it

```
import pandas as pd
import numpy as np

tx_raw = pd.DataFrame({
    "order_id":      ["O1001","O1001","O1002","O1003","O1004",
                      "O1004","O1005","O1006","O1006","O1007",
                      "O1008","O1009","O1009","O1010"],
    "order_line_id": ["L1","L2","L3","L4","L5","L6","L7","L8",
                      "L8","L9","L10","L11","L12","L13"],
    "customer_id":   ["C01","C01","C02","C03","C04","C04","C05",
                      "C06","C06","C07","C08","C09","C09","C10"],
    "product_id":    ["P010","P022","P031","P010","P044","P051",
                      "P022","P063","P063","P031","P044","P075",
                      "P075","P010"],
    "store_id":      ["ONLINE","ONLINE","S01","APP","S07","S07",
                      "ONLINE","S01","S01","ONLINE","S09","ONLINE",
                      "ONLINE","APP"],
    "channel":       ["Online","Online","Store","App","Store",
                      "Store","online","Store","Store","web",
                      "Store","Online","Online","App"],
    "order_date":    ["2025-11-03","2025-11-03","2025-11-04",
                      "2025-11-04","2025-11-05","2025-11-05",
                      "2025-11-06","2026-01-06","2026-01-06",
                      "2025-11-07","2025-11-08","2025-11-08",
                      "2025-11-09","2024-06-15"],
    "quantity":      [1, 2, 1, 1, 3, 2, 1, 1, 1, 2, 999, 1, 1, 1],
    "unit_price":    ["$38.00","$29.00","$52.00","$38.00","$74.00",
                      "$81.00","$29.00","$45.00","$45.00","$52.00",
                      "$36.00","$240.00","$68.00","$38.00"],
    "discount_pct":  [0.0, 0.20, 0.0, np.nan, 0.0, 0.0, 0.20,
                      0.0, 0.0, 0.30, 0.0, 0.0, 0.0, 0.0],
    "campaign_id":   ["CMP04","CMP04", None, "CMP11", None, None,
                      "CMP04", None, None, "CMP09", None, None,
                      None, "CMP11"]
})

customers_mini = pd.DataFrame({
    "customer_id": ["C01","C02","C03","C04","C05",
                    "C06","C07","C08","C09","C10"],
    "signup_date": pd.to_datetime([
        "2024-05-12","2024-10-01","2025-09-20","2025-07-30",
        "2024-08-15","2025-12-28","2025-10-05","2024-07-01",
        "2025-06-22","2025-01-10"])
})

# The raw object is evidence. Never transform it in place;
# every preparation step below runs on the working copy, tx.
tx = tx_raw.copy(deep=True)

tx_raw.shape
```

Expected output: (14, 11) — fourteen order lines, eleven columns. Log entry one: baseline rows = 14.

Input: two literal frames, the fourteen-line extract and the ten-customer dimension table reused from Lab 3.1, so the signup dates are already familiar. Transformation: none yet, apart

from the deep copy that separates the evidence from the workspace. Output: the baseline the whole log is checked against. Every later count in this lab is a claim about how far the file has moved from (14, 11), and tx_raw is what makes that claim auditable rather than remembered. Now run the inspection routines of Section 4.3 in order — dtypes, head, info, describe — predicting each output before executing.

Code 4.2. Inspect before touching

```
# A notebook renders only a cell's last expression, so each
# inspection result is displayed explicitly.
from IPython.display import display

display(tx.dtypes)
display(tx.head(14))
tx.info()
display(tx.select_dtypes(include="number").describe())
```

Expected output, worth predicting in detail: unit_price reports object, not a number — the currency strings have made the entire column text, so the numeric-only describe excludes it entirely, an absence that is itself a finding. quantity reports a maximum of 999. discount_pct reports 13 non-null values of 14. The head reveals the channel variants and the repeated L8 line.

Input: the untouched working copy. Transformation: none — inspection is a read-only stage, which is why it can be run before any prediction has been risked. Output: four diagnostic views, each displayed explicitly. A notebook automatically renders only the last expression in a cell, so a cell listing four routines shows one of them; display() is used here, and wherever a later cell reports more than one result, so that the outputs the prose asks students to read are the outputs students actually see. Taken together the four produce the written diagnosis. For the miniature it reads: one type defect (unit_price), one label defect (channel), at least one exact duplicate (the repeated L8 row), a quantity sentinel, one missing discount_pct, and — pending checks — possible near-duplicates, an out-of-band price hiding inside the text column, and date inconsistencies. The pipeline now runs in the default order of Section 4.2: types and labels, duplicates, impossible values, missing values, derivations.

Code 4.3. Step 1 — coerce unit_price from currency strings

```
before_na = tx["unit_price"].isna().sum()

# Normalize to text first: a real extract may mix strings with
# values that already arrived numeric.
price_text = tx["unit_price"].astype("string")

clean_price = price_text.str.replace(r"[$,]", "", regex=True)

tx["unit_price"] = pd.to_numeric(clean_price, errors="coerce")

after_na = tx["unit_price"].isna().sum()
print(before_na, "→", after_na, "| rows:", len(tx))
print("max price:", tx["unit_price"].max())
```

Expected output: 0 → 0 | rows: 14, and a maximum price of 240.0.

Input: a text column holding fourteen currency strings. Transformation: normalize to the string dtype, strip the dollar signs and any thousands separators, then convert with errors="coerce" so that failures become missing values the code can count rather than exceptions that stop the notebook. Output: a numeric column, an unchanged row count, and a maximum that now announces the out-of-band price the object dtype was hiding. Confirm that tx_raw["unit_price"] still reads "$38.00" in its first row: the repair happened on the copy.

VERIFICATION CHECK

Before you run: every one of the fourteen strings is a dollar sign followed by a clean number, so all fourteen should convert. Predict missing values in unit_price going from 0 to 0, a row count that stays at 14, and — reading the raw strings — a maximum of $240.00 once the column is numeric.

After you run: before_na and after_na both report 0, len(tx) reports 14, and the maximum is 240.0.

Investigate if: after_na comes back greater than zero. That means errors="coerce" has silently nulled a value the prediction said would convert; find the offending string in tx_raw before proceeding, because the raw column is still intact.

Code 4.4. Step 2 — standardize the channel labels

```
print("Before standardization")
display(tx["channel"].value_counts(dropna=False))

channel_map = {
    "Online": "Online",
    "online": "Online",
    "web": "Online",
    "Store": "Store",
    "App": "App",
}
tx["channel"] = tx["channel"].map(channel_map)

print("After standardization")
display(tx["channel"].value_counts(dropna=False))
```

Expected output: before — five labels (Store 6, Online 4, App 2, online 1, web 1); after — three labels (Online 6, Store 6, App 2). Row count unchanged at 14, and no missing values in channel.

Input: a categorical column carrying five spellings of three channels. Transformation: an explicit mapping from every observed variant to its canonical label, built from the value counts rather than from memory. Output: two labeled tallies, before and after, because the comparison between them is the verification — three canonical labels, the same fourteen rows, and no new missing values. A missing value after mapping would mean the enumeration was incomplete — map returns NaN for any key it was not given — which is precisely why the value counts are read before the map is written. The mapping decision itself, that "web" is the Online channel and not a distinct legacy channel, is recorded in the log as a judgment call, with its source (in the course setting, the data dictionary; at StyleCraft, the e-commerce platform owner).

Code 4.5. Step 3 — remove exact duplicates, then near-duplicates

```
exact = tx.duplicated().sum()
print("exact duplicates:", exact)
tx = tx.drop_duplicates().copy()
print("rows:", len(tx))

identity = ["order_id","customer_id","product_id",
            "quantity","unit_price"]
near = tx.duplicated(subset=identity).sum()
print("near-duplicates on identity columns:", near)
tx = tx.drop_duplicates(subset=identity).copy()
print("rows:", len(tx))
```

Expected output: exact duplicates: 1, rows: 13; near-duplicates: 0, rows: 13.

Input: fourteen rows with canonical labels and numeric prices — the order matters, because a duplicate check run before Step 1 would have compared "$45.00" to "$45.00" as text and a check run before Step 2 would have treated "online" and "Online" as different events. Transformation: drop rows identical in every column, then drop rows identical on the declared

identity columns. Output: thirteen rows and a near-duplicate count of zero — which is the correct result for this rule and a deliberately uncomfortable one, as the Verification Check explains.

VERIFICATION CHECK

Before you run: two predictions. First, exactly one row is a full copy of another (the repeated L8 line), so exact = 1 and the row count falls 14 → 13. Second, order O1009 appears twice with the same customer and product but different order_line_id, order_date, and — check the raw strings — different prices ($240.00 and $68.00). Ask whether the identity rule above catches it, and write the answer down before running.

After you run: exact = 1, rows 13; near = 0, rows still 13. The identity rule does not catch the O1009 pair, because unit_price is in the identity list and the prices differ.

Investigate if: near comes back greater than zero, or the row count falls below 13. Either means the identity columns are catching events the rule was not meant to catch. Record the lesson either way: near-duplicate detection and error correction interact, and the O1009 pair is resolved by Step 4's domain-limit check instead. Exercise 4.7 revisits this trap.

Code 4.6. Step 4 — impossible values, a validated join, and dates

```
# Domain limits from the data dictionary: prices $24-$90,
# plausible basket quantities.
bad_price = tx[(tx["unit_price"] < 24) | (tx["unit_price"] > 90)]
bad_qty   = tx[tx["quantity"] > 20]
print(bad_price[["order_id","order_line_id","unit_price"]])
print(bad_qty[["order_id","order_line_id","quantity"]])

# Cross-table date check. State the expected cardinality and
# make the code enforce it: one customer row per transaction.
tx["order_date"] = pd.to_datetime(tx["order_date"])
rows_before_join = len(tx)

tx = tx.merge(
    customers_mini,
    on="customer_id",
    how="left",
    validate="many_to_one",
    indicator=True
)

assert len(tx) == rows_before_join, "the join changed the row count"
print(tx["_merge"].value_counts())
print("unmatched customers:", tx["signup_date"].isna().sum())
tx = tx.drop(columns="_merge")

bad_dates = tx[tx["order_date"] < tx["signup_date"]]
print(bad_dates[["order_id","customer_id",
                 "order_date","signup_date"]])
```

Expected output: one out-of-band price (O1009's $240.00 line, L11), one sentinel quantity (the 999 on O1008, L10), a merge indicator reporting 13 both and 0 left_only, zero unmatched

customers, and one impossible date — O1010, ordered 2024-06-15 by C10, who signed up 2025-01-10.

Input: the deduplicated thirteen-row frame and the customer dimension. Transformation: three diagnostic filters and one join. The join is where this cell earns its keep. Counting unmatched rows after a merge, as an earlier draft of this lab did, detects a customer who is missing from the dimension table; it does not detect a customer who appears in it twice. A duplicated key silently multiplies transaction rows, inflating revenue while producing no error and no unmatched row to count — the wrong-join-key failure of Section 4.10 in its most flattering disguise. Declaring validate="many_to_one" makes pandas raise if the right-hand key is not unique, and the assertion on the row count states the same expectation in the log's own vocabulary. Output: the flagged rows, an explicit matched/unmatched tally from the indicator column, and a row count that has provably not moved. The indicator column is dropped once it has been read, because a verification artifact does not belong in the certified table.

Each flagged row now gets classified per Section 4.5, and the classifications are the log's judgment entries. The $240.00 line, sitting beside its $68.00 twin on the same order, is resolved as the erroneous member of a suspected re-export pair. Note what the file alone can and cannot establish here. The domain limit establishes that $240.00 is out of band and therefore wrong; it does not establish what the right value was, and $240.00 is not a decimal shift of $68.00, so the two lines cannot be reconciled by arithmetic. The defect catalog — or, at StyleCraft, the source-system record — identifies L12 as the valid line, which is what licenses the judgment: exclude L11 and retain the plausible line. Where no such external record exists, the defensible move is not to guess which line is right but to set the out-of-band value to missing and say so. The 999 quantity has no corroborating basket — it is treated as a sentinel and set to missing, converting it into Section 4.4's problem with a known mechanism. The impossible date cannot be repaired from the file alone; the ground rule is documented exclusion, so the O1010 line is removed under a written rule. Note what the miniature keeps: the big-but-legitimate basket on O1004 (three units at $74 plus two at $81) survives every rule, because nothing about it is an error — it is the occasionwear tail, and deleting it would have deleted the phenomenon. The next cell applies the three classifications; predict the row counts before running it.

Code 4.7. Step 4b — apply the classifications

```
# Judgment 1: the defect catalog identifies L11 as the
# erroneous member of the O1009 re-export pair.
tx = tx[tx["order_line_id"] != "L11"].copy()
print("rows after removing the erroneous re-export line:", len(tx))

# Judgment 2: the 999 quantity is a sentinel. Declare the
# nullable integer type first, so a count stays a count.
tx["quantity"] = tx["quantity"].astype("Int64")
tx.loc[tx["quantity"] > 20, "quantity"] = pd.NA
print("quantity dtype:", tx["quantity"].dtype,
      "| missing quantities:", tx["quantity"].isna().sum())

# Judgment 3: documented exclusion of the impossible-date line.
tx = tx[~(tx["order_date"] < tx["signup_date"])].copy()
print("rows after the date exclusion:", len(tx))
```

Expected output: rows fall 13 → 12 after the re-export removal and 12 → 11 after the date exclusion; quantity reports dtype Int64 with one missing value. Setting the sentinel to missing changes no row count, only the missing-value tally. Three rows of the log now carry judgment flags.

Input: thirteen rows carrying three classified defects. Transformation: one row removed under an error rule, one value set to missing under a sentinel rule, one row removed under an exclusion rule. Output: eleven rows, each departure explained. The type declaration deserves a sentence of its own. Assigning a missing value into an ordinary integer column promotes the column to floating point, and quantities then display as 1.0 and 2.0 — a count that no longer looks like a count. Pandas' nullable Int64 type exists for integer data that may contain missing values (pandas development team, 2026b), and declaring it before the assignment preserves what the column measures. This is the Section 4.7 principle applied to the cleaning's own side effects: a repair that changes a column's claim about its quantity is a defect the repair introduced.

Code 4.8. Step 5 — treat the remaining missing values and derive

```
tx["quantity_missing"] = tx["quantity"].isna()
tx["discount_missing"] = tx["discount_pct"].isna()
tx["discount_pct"] = tx["discount_pct"].fillna(0.0)

tx["line_revenue"] = (tx["quantity"] * tx["unit_price"]
                      * (1 - tx["discount_pct"]))

print("rows:", len(tx))
print("computable lines:", tx["line_revenue"].notna().sum())
print("known revenue:", round(tx["line_revenue"].sum(), 2))
```

Expected output: rows: 11, computable lines: 10, known revenue: 767.4.

Input: eleven surviving rows, one with an unresolved quantity and one that had a missing discount. Transformation: two flag columns, one justified fill, and the recomputation of line_revenue from its repaired inputs. Output: eleven rows and a revenue figure that is not the

revenue of eleven rows. Pandas' sum excludes missing values by default (pandas development team, 2026c), so $767.40 is the total of the ten computable lines; the line whose quantity became missing in Step 4b contributes nothing and is not counted as zero either. Every student should reproduce $767.40 with a calculator from the ten surviving computable lines.

Two decisions in this cell are judgment entries rather than arithmetic. Filling a missing discount_pct with 0 asserts that the line was sold at full price, which is a business judgment — the modal case, per the dictionary — and not a statistical necessity; the flag column preserves the ability to test its sensitivity later. The missing campaign_id values are left exactly as they are: Section 4.4's two-kinds-of-null problem cannot be resolved from inside the file, the dictionary records the ambiguity, and imputing an attribution would manufacture marketing performance. One step remains, and it is the step that separates a cleaned file from a certified one. Eleven rows survive, every departure from the raw file is explained, and the arithmetic is correct — and yet one surviving row carries a revenue value that is not merely unknown but undecided, because the sentinel quantity was converted to missing rather than resolved. Section 4.2.1 defined certification as a threshold: a file is certified for a stated use when no quantity that use depends on is left unresolved. Revenue analysis depends on line_revenue. The gate below makes the threshold executable rather than rhetorical.

Code 4.9. Step 6 — the certification gate

```
unresolved = int(tx["line_revenue"].isna().sum())
known_revenue = tx["line_revenue"].sum()

print("surviving rows:", len(tx))
print("rows with unresolved revenue:", unresolved)
print("known revenue across computable lines:",
      round(known_revenue, 2))

if unresolved == 0:
    print("STATUS: certified for revenue analysis.")
else:
    print("STATUS: provisionally cleaned —", unresolved,
          "row(s) carry unresolved revenue.")
```

Expected output: surviving rows: 11, rows with unresolved revenue: 1, known revenue across computable lines: 767.4, and the status line reporting provisionally cleaned.

Input: the prepared eleven-row table. Transformation: none — the gate changes nothing, which is the point of a gate. Output: a verdict. Part A does not certify. It closes as provisionally cleaned, with one unresolved quantity, and reports known revenue across ten computable lines of $767.40. The distinction is not pedantry. A table offered as certified is a table whose totals a CFO may use without asking further questions, and $767.40 is not the miniature's revenue; it is the revenue of the part of the miniature that could be computed. Resolving the eleventh line requires something the file does not contain — the true quantity from the source system, or a documented decision to exclude the line — and until one of those arrives, the honest label is the

provisional one. Close Part A's log accordingly: 14 rows in, 11 out, every departure explained, one quantity outstanding, and revenue computable for ten of eleven surviving lines.

In Part B, where the full file must support the expansion review, the same gate is written as an assertion rather than a status line, because the answer key admits no unresolved revenue: assert transactions_clean[“line_revenue”].isna().sum() == 0, with a message naming certification as the thing being blocked. A pipeline that cannot pass that line has not finished, however clean its output looks.

4.11.2 Lab 4.1, Part B: The Full File

Part B applies the same pipeline to StyleCraft's actual transactions_raw file, available with customers, products, and stores from Brightspace and the companion repository. The defects are the same families at realistic scale, and the file ships with two anchors the miniature could not offer: the transactions_clean answer key, against which the final table is compared, and the known totals (row counts, distinct orders, total revenue) that transactions_clean implies.

Work through the pipeline in the same order, with four additions the scale demands. First, copy the raw frame before the first transformation, exactly as Code 4.1 does, and keep it in memory to the end. Second, validate every join. Part B joins transactions to customers, to products, and to stores, and each of the three is a many-to-one relationship that the code should declare and enforce with the validate and indicator arguments of Code 4.6, bracketed by a row count before and an assertion after. Extend the impossible-date check to stores by joining on store_id and flagging order dates before the store's opening_date, remembering that ONLINE and APP are reserved store_id values with no opening date — predict how the join treats them before running it. Third, let an AI assistant draft each step's code per the AI in Practice pattern, and keep its predictions and yours side by side in the log; at full scale, at least one delegated step will disagree with your prediction, and finding out why is the assignment. Fourth, close the log with the certification entries: the cleaned file's row count, distinct-order count, and total revenue against transactions_clean's; the certification gate of Code 4.9 written as an assertion; and the reconciliation that opened the chapter — how much of the raw file's excess revenue the removed duplicates and corrected errors account for.

VERIFICATION CHECK

Before you run: predict, for each of the three dimension joins, the row count afterward (unchanged), the number of unmatched rows, and which reserved store_id values you expect to appear as unmatched by design. Predict the final row count, distinct-order count, and total revenue before comparing them to transactions_clean.

After you run: every join leaves the transaction row count unchanged; the unmatched counts match your predictions or are explained; the certification assertion on unresolved revenue passes; and the final totals reconcile with the answer key.

Investigate if: any difference from the key is not attributable to a logged decision. The comparison against transactions_clean is graded self-verification, in keeping with the book's known-truth verification thread: the answer key exists so the pipeline can be checked, not copied. Matching totals do not require matching code — two defensible pipelines can differ in judgment entries — but an unexplained difference of even one row means a silent step ran somewhere. Find it.

4.11.3 Lab 4.2: First Exploration of the Cleaned File

Lab 4.2 explores transactions_clean (or your certified equivalent), practicing Section 4.9's questioning discipline. Charts here use the basic matplotlib patterns from Appendix A; design is not the point, questions are — the theory of chart selection and encoding is Chapter 12's. Four worked cells follow, each beginning with a written prediction and ending with the anomalies the picture raises. Run them on the full cleaned file; the miniature is too small to have a distribution worth reading.

The lab opens with a deliberate grain change, because every chart after it is drawn at order grain rather than line grain and the difference is the whole of Section 4.8. The aggregation below also carries two verifications and one deliberate choice worth reading closely before running the cell.

Code 4.10. Aggregate order lines to order grain

```
# Precondition 1: this cell claims a certified input.
assert transactions_clean["line_revenue"].isna().sum() == 0, (
    "order aggregation requires a table certified "
    "for revenue analysis"
)

# Precondition 2: "first" is only honest if the attribute is
# constant within the order. Check, do not assume.
for column in ["store_type", "channel"]:
    constant = (
        transactions_clean
        .groupby("order_id", dropna=False)[column]
        .nunique(dropna=False)
        .le(1)
        .all()
    )
    assert constant, f"an order carries multiple {column} values"

# One row will equal one order.
order_summary = (
    transactions_clean
    .groupby("order_id", as_index=False, dropna=False)
    .agg(
        order_revenue=("line_revenue",
                       lambda s: s.sum(min_count=1)),
        lines=("line_revenue", "size"),
        unresolved_lines=("line_revenue",
                          lambda s: s.isna().sum()),
        store_type=("store_type", "first"),
        channel=("channel", "first"),
    )
)

# Verification 1: one row per distinct order.
assert len(order_summary) == \
    transactions_clean["order_id"].nunique(dropna=False)

# Verification 2: the grain change preserves the total.
assert np.isclose(
    order_summary["order_revenue"].sum(),
    transactions_clean["line_revenue"].sum()
)

print("orders:", len(order_summary))
print("order revenue total:",
      round(order_summary["order_revenue"].sum(), 2))
print("orders with unresolved lines:",
      int((order_summary["unresolved_lines"] > 0).sum()))
```

Expected output: an order count equal to the distinct order_id count, an order-grain revenue total equal to the line-grain total to the cent, and zero orders carrying unresolved lines. All four assertions pass silently.

Input: the certified line-grain table. Transformation: rows are collected by order_id and summarized, with each column given the summary its meaning deserves — revenue sums, lines

count, the dimension attributes take the first value within the order. Output: an order-grain table and four proofs that the grain change was faithful. The assertions are the cell, and each states an expectation the prose would otherwise leave as an assumption. The first makes the input requirement executable: this cell says it consumes a certified table, and a certified table has no unresolved revenue, so the claim is checked rather than trusted. The second guards the word first. Taking the first store_type in a group is honest only if every line in the order carries the same one; if a contaminated order held two, the aggregation would silently keep whichever appeared first and the order would be filed under a store type it only partly belongs to. The remaining two assertions are the grain-change identities of Section 4.8: one row per distinct order, and a total preserved to the cent.

Two arguments in the aggregation deserve the same attention. The dropna=False argument makes the treatment of missing group keys explicit rather than default, because pandas otherwise drops them without comment (pandas development team, 2026a) and orders would disappear silently. The min_count=1 argument keeps an order whose lines are all unresolved as missing instead of collapsing it to $0.00 — the trap Section 4.8 named, in which an unknown quantity is quietly promoted to a confident zero that no total will catch, because zero adds nothing to a sum. On a certified input the first assertion makes that trap unreachable, and the pattern is kept anyway, for two reasons: it is the habit that protects the analyst on the day the input is provisional rather than certified — Part A's table is exactly such a case — and the unresolved_lines column carries the certification question up to the new grain, so that the order table can be gated the same way the line table was.

VERIFICATION CHECK

Before you run: predict the number of orders (the distinct order_id count from your certified file), predict that the order-grain revenue total will equal the line-grain total exactly, and state — before the code tells you — whether you expect any order to contain more than one store_type or channel, and why.

After you run: all four assertions pass silently, and the printed order count and revenue total match the predictions.

Investigate if: any assertion fails. A failed certification check means the input is provisional and the gate of Code 4.9 was never cleared. A failed constancy check means an order spans two store types or channels, which is a defect in the upstream join or a real business case the aggregation must handle deliberately rather than by taking whichever value came first. An order count above the distinct-order count usually means a duplicated key in an upstream join, and a revenue total below the line-grain total usually means a group key containing missing values. A total that matches while some order reports $0.00 revenue means min_count was omitted and unresolved lines were summed as zero.

With the grain settled, the first question is the shape of the order-value distribution. Write the prediction first: from the business story, order revenue should be right-skewed and bounded below, with a long tail produced by occasionwear baskets, and — because StyleCraft was

designed with two customer populations — a second hump would not be a surprise so much as a confirmation.

Code 4.11. Histogram of order revenue

```
import matplotlib.pyplot as plt

revenue = order_summary["order_revenue"].dropna()

fig, ax = plt.subplots(figsize=(7, 4))
ax.hist(revenue, bins=40, edgecolor="white")
ax.set_xlabel("Order revenue ($)")
ax.set_ylabel("Number of orders")
ax.set_title("Distribution of order revenue")
plt.show()

print(revenue.describe().round(2))
print("orders dropped as unresolved:",
      len(order_summary) - len(revenue))
```

Expected output: a right-skewed histogram concentrated at the low end with a thinning tail to the right, a mean above the median in the describe output, and a count of dropped orders equal to the unresolved-order count from Code 4.10.

Input: the order-grain revenue column. Transformation: values are binned and counted; the dropna is stated explicitly and its cost is printed, because a chart that silently omits rows is the histogram version of a silent row drop. Output: a picture, a five-number summary to read alongside it, and an accounting of what the picture excludes. Read the picture in the order Section 4.9.2 prescribes: shape first (is the skew there, and does the mean sit above the median as skew implies?), then spikes (a bar standing above its neighbors at a suspiciously round value is a sentinel or a cap that survived cleaning), then gaps and orphans (an empty range, or a lone bar far to the right, is an outlier rule made visible). Each anomaly becomes a log entry with a resolution: a defect the cleaning missed, a documented business fact, or a finding. A second hump belongs in the third category — it is the two designed customer populations showing themselves, and Chapter 6 will name them.

The histogram describes the file as a whole. The next question is comparative, and it is the one the expansion review actually asks: do suburban stores generate larger orders than urban stores, and is the difference in the middle of the distribution or in its tail? Predict the answer before drawing it — which group has the higher median, and which has the longer tail — because a prediction written down is what turns the picture into evidence rather than decoration.

Code 4.12. Grouped box plots of order revenue

```
def revenue_by(frame, column):
    """Return (labels, arrays) for a box plot grouped by column."""
    labels, groups = [], []
    for key, part in frame.groupby(column, dropna=False):
        labels.append(str(key))
        groups.append(part["order_revenue"].dropna().to_numpy())
    return labels, groups

for column, title in [("store_type", "store type"),
                      ("channel", "channel")]:
    labels, groups = revenue_by(order_summary, column)

    fig, ax = plt.subplots(figsize=(7, 4))
    ax.boxplot(groups, whis=1.5)
    ax.set_xticks(range(1, len(labels) + 1))
    ax.set_xticklabels(labels)
    ax.set_ylabel("Order revenue ($)")
    ax.set_title("Order revenue by " + title)
    plt.show()

    print(order_summary.groupby(column, dropna=False)
          ["order_revenue"]
          .agg(["count", "median", "mean", "max"]).round(2))
```

Expected output: two box plots, one per grouping, each accompanied by a table of group counts, medians, means, and maxima. The suburban group should show both a higher median and a longer upper tail than the urban group; every group's mean should sit above its median.

Input: the order-grain table and two grouping columns. Transformation: orders are split into groups, each group's quartiles are computed, and the whiskers are drawn under the default Tukey convention — extending to the most extreme observations lying within 1.5 interquartile ranges of the quartiles, with anything beyond drawn as an individual point (Matplotlib Development Team, 2026). Output: a comparison that a pair of averages would flatten, plus the numbers behind it. Read the boxes for medians that differ, for boxes of very different heights (one group is more heterogeneous than another), and for dense swarms of plotted points above the upper whisker. That last pattern is the one this chapter has been arguing about: those points are the occasionwear baskets, they are legitimate extremes rather than defects, and an analyst who had trimmed them in Section 4.5 would now be looking at a picture with the expansion strategy's best evidence removed from it.

The final chart returns to line grain and to arithmetic. A scatterplot of quantity against line revenue should show a fan of points organized into bands, because line revenue is quantity multiplied by a price drawn from a bounded catalog. Predict the structure before drawing it: how many bands, and where should the fan's upper edge sit for a basket of three units?

Code 4.13. Scatterplot of quantity against line revenue

```
lines = transactions_clean.dropna(
    subset=["quantity", "line_revenue"]
)

fig, ax = plt.subplots(figsize=(7, 4))
ax.scatter(lines["quantity"], lines["line_revenue"],
           s=8, alpha=0.3)
ax.set_xlabel("Quantity (units on the line)")
ax.set_ylabel("Line revenue ($)")
ax.set_title("Line revenue against quantity")
plt.show()

# Anything above the catalog ceiling is a specific row that
# owes a specific explanation.
ceiling = lines["quantity"] * 90
strays = lines[lines["line_revenue"] > ceiling]
print("lines above the catalog ceiling:", len(strays))
print(strays[["order_id", "order_line_id", "quantity",
              "unit_price", "line_revenue"]].head(10))
```

Expected output: a fan of points rising with quantity, its upper edge tracking $90 per unit and its lower edge falling below $24 per unit where discounts apply, and — on a correctly cleaned file — zero lines above the catalog ceiling.

Input: the certified line-grain table, restricted to lines with both a quantity and a revenue. Transformation: one point per line, with transparency so that dense bands remain readable. Output: a structure and a test. The structure is the expected one — revenue rising with quantity along bands implied by the $24-to-$90 catalog. The test is the printed stray count, which converts “anything outside the fan is a specific row with a specific explanation owed” from an instruction into a number. A stray above the ceiling at quantity one is an arithmetic defect that survived cleaning, most likely a decimal error of the kind Section 4.5 describes; a cluster of strays sharing a store or a channel is a source-system problem rather than a row problem. Either way the finding goes in the log, and each chart's question, prediction, and resolution goes in the notebook in words — the written trail is what Chapter 5 will pick up as the difference between an observation and an insight.

4.12 Marketing Interpretation and Managerial Insight

The labs produced a certified table, a completed log, and a first exploration. This section returns to the conference room, because the ten days end in a meeting, and asks what the preparation work means in decision language.

The headline the analytics lead can now deliver is not a number but a warrant for numbers: the table of record reconciles to finance's quarterly total, and every difference between it and the raw extract is a written, defensible decision — this many duplicated lines removed, these erroneous values corrected or excluded under stated rules, these gaps flagged rather than papered over. The reconciliation gap has an explanation instead of a shrug. That warrant changes the

texture of the expansion review before a single analysis is presented: the same-age cohort comparison and the store-economics summary will be computed on rows both the CFO and the CRM manager have agreed to trust, which means the meeting can argue about the business instead of about the file — the preparation-stage counterpart of what Chapter 3 said good measurement buys.

Now the wrong managerial reading, because there is a tempting one. A stakeholder scanning the log summary might conclude: “The raw feed overstated revenue; the data team fixed it; so the expansion numbers were too optimistic and the pause should continue.” Every clause outruns the evidence. The duplicates inflated total revenue, but nothing yet says they inflated the suburban stores' revenue disproportionately — duplication that hit all channels evenly changes levels, not comparisons, and it is the comparisons the expansion decision turns on. Nor does a cleaner file settle the direction of any conclusion: the corrected table could make the suburban economics look better (if the removed 999-quantity sentinels had been landing in store rows) or worse. The correction to offer the room is precise: the cleaning changed how much the numbers can be trusted, not yet what the numbers say, and the “what” is next week's work — Chapter 5's — now standing on certified ground. The discipline of separating those two claims is, in miniature, the discipline of the whole course.

Two more managerial translations deserve the room's attention. The first is the judgment entries. The log contains a handful of decisions that were choices, not arithmetic — “web” mapped to Online, missing discounts read as full price with a flag, the impossible-date lines excluded rather than repaired. A manager does not need to review pandas; a manager does need to know that these choices exist, that each has a stated rationale, and that each can be revisited if a stakeholder's knowledge contradicts it. Surfacing them takes three minutes of the meeting and buys something valuable: when a choice is later questioned — and one always is — the answer is a log entry, not a reconstruction from memory.

The second is the language of certification itself. “Certified” is not a synonym for “finished,” and an analyst who uses it loosely will eventually be asked to defend a total that quietly excluded rows nobody mentioned. A table is certified for a stated use, and the statement has a scope: this table supports revenue analysis at order and line grain for this window, with these exclusions, and it does not yet support any analysis that depends on the quantities still unresolved. Part A of the lab makes the point at fourteen rows — eleven rows survive, ten of them carry computable revenue, and the honest label is provisional. Saying so costs a sentence in the meeting. Not saying so costs the analyst's credibility the first time someone recomputes the total and finds a different number.

4.13 Business Analytics in Practice

The chapter so far has practiced preparation on StyleCraft. This section steps out of the fiction and looks at how the same work appears in industry — how much of the job it is, how expensively it fails, and how organizations have tried to industrialize it.

4.13.1 How Much of the Job Preparation Actually Is

The first thing to know is how large this work looms in real analyst time. The folk statistic — that data scientists spend roughly 80 percent of their time finding, cleaning, and organizing data — has circulated for decades, and the precise figure is best treated as a trope rather than a measurement: estimates vary by survey and by decade, and one widely read practitioner survey put data loading and cleansing at roughly 45 percent of working time rather than 80 (Anaconda, 2020). What the estimates agree on, however, is the ordering: preparation occupies more analyst time than modeling does, a pattern the data-mining literature documented well before modern tooling (Dasu & Johnson, 2003).

What is instructive is how stable the finding has remained across tool generations. Spreadsheets gave way to SQL, SQL to pandas, pandas to AI copilots, and each generation was accompanied by predictions that cleaning drudgery was about to disappear; each generation instead made it faster to produce more data, from more systems, with more inconsistencies. AI assistance is the newest chapter in that story, and the early pattern matches this guide's stance: assistants compress the mechanical hours substantially, while the diagnosis and judgment hours — is this null informative, is this extreme legitimate, which columns define identity — remain stubbornly human, because they were never coding hours in the first place. Analysts who resent preparation as a tax on "real" analytics tend to be surprised by this; analysts who understand preparation as the first analysis, in this chapter's sense, are the ones organizations learn to trust with the file of record.

4.13.2 The Revenue Decline That Was a Tracking Tag

The second vignette is about how silently preparation-level defects can masquerade as business events. A mid-market e-commerce retailer — the composite is generic, but the pattern is one of the most repeated war stories in analytics — opened a quarterly review to a chart showing online revenue in steady decline over six weeks. The room did what rooms do: it generated explanations. A competitor's promotion, a weak product drop, rising ad costs; one executive proposed reallocating budget to paid search that afternoon. The analyst who eventually traced the decline found no customer behavior in it at all: a website tracking-tag update, deployed at exactly the start of the decline, had stopped recording a slice of completed orders, and the "revenue decline" was a measurement outage wearing a trend costume.

The story earns its place in this chapter for two reasons. The defect was invisible to every within-file check — the file was internally consistent, merely incomplete — and would have been caught in a day by the humble discipline of Section 4.3, reconciliation against a known external total (the payment processor's settlement figures never dipped). And the near-miss was a decision, not a dashboard: budget was hours from moving on the strength of a defect. The practice lesson is the chapter's thesis with money attached — an undiagnosed data defect does not merely mislead a chart; it manufactures a business narrative, and rooms act on narratives.

4.13.3 Industrializing the Log

The third vignette concerns industrialization. In organizations that run analytics at scale, the pipeline of Section 4.2 does not live in one analyst's notebook; it lives in engineered data pipelines, run on schedules, with the verification log's logic converted into automated tests. Modern transformation frameworks — dbt is the best-known exemplar of the pattern — let teams attach declarative quality tests to every table in the warehouse: this key is never null, this key is unique, this value stays within an accepted set, this child table's keys all exist in the parent, and whatever additional business rules the team writes as custom tests (dbt Labs, n.d.-a).

Every scheduled run can execute the tests, and organizations may configure a failure to stop a build and alert an owner before a defective table reaches a dashboard. The qualification matters, because the framework supplies the assertions rather than the organizational response: tests carry a severity that determines whether a failure warns or errors, and what happens next — whether a build halts, whether anyone is paged — depends on how the deployment and monitoring around the framework are configured (dbt Labs, n.d.-b). Automation makes the checks cheap and repeatable; someone still has to decide that a failure is worth waking a person for.

Students should recognize the tests immediately — they are Section 4.3's inspection routines, Section 4.5's domain limits, and Section 4.6's cardinality expectations, written once and run forever, and the “known totals” reconciliation of the opening case becomes, in this world, a permanent test rather than a heroic catch. What industrialization does not remove is the analyst's role at the edges of the automation: someone must decide what the tests should assert (the domain limits come from business knowledge, not from the warehouse), someone must adjudicate the failures (is the out-of-band value an error or a policy change?), and someone must own the judgment entries no test can make. Automation, in other words, industrializes the log's arithmetic and leaves its editorial content exactly where this chapter put it — with the analyst of record.

4.13.4 In Your First Analyst Job

In your first analyst job, these vignettes compress into a single expectation worth internalizing early. You will spend more of your first year preparing data than modeling it; the preparation will periodically be where the real finding lives; and the organization's memory of what was removed, changed, and assumed will reside wherever you write it down — which is to say, if you do not keep the log, it resides nowhere. Cleaning decisions are analytic decisions, and the documentation is not overhead on the work; it is the part of the work that separates the analyst from the liability when a number is challenged six months later. The analysts who advance are rarely the ones who cleaned fastest. They are the ones whose files the organization learned it could certify.

4.14 Ethics, Cleaning Decisions, and the Documented Exclusion

The Business Analytics in Practice section described cleaning as work organizations learn to trust. This section examines the ethical weight of that trust, extending the guide's running discussions (Sections 1.13, 2.12, and 3.13) from data use, problem framing, and measurement design to the cleaning step itself. The chapter's angle is specific: cleaning is silent editorial power, and its ethical discipline is the documented exclusion.

Begin with why the power is editorial. Media scholars have made the point compactly: “raw data” is something of an oxymoron, because data arrives already shaped by collection choices, and every subsequent act of tidying shapes it further (Gitelman, 2013). Digital humanities scholars pressed the argument directly at cleaning, observing that the very word “cleaning” flatters the work — it implies removing grime from a fixed underlying truth, when what actually happens is that someone decides which variations are noise and which are signal, and the decisions are then hidden inside a file that looks pristine (Rawson & Muñoz, 2019). This chapter's vocabulary makes the same point without the theory: the judgment entries in the verification log — what counts as a duplicate, which extremes are errors, what a missing value is assumed to mean, which rows are out of scope — are decisions about whose behavior the certified table represents. They are small, individually defensible, and cumulatively they determine what the expansion review will see.

The ethical risk concentrates where the editing intersects people unevenly. Consider three of the lab's own decisions through that lens. Dropping rows with missing age_band looks hygienic until Section 4.4's mechanism question is asked: if in-store customers skip the optional question more often, the drop disproportionately removes store shoppers — the suburban expansion's own customers — from every age-based analysis, and the review's picture of “who shops the new stores” is quietly reweighted toward people who shop online. Excluding the impossible-date rows is defensible, but the rule must be blind: an exclusion rule applied more diligently to rows that hurt a preferred conclusion than to rows that help it is not cleaning, it is curation. And the treatment of extremes carries a distributional thumb: winsorizing or trimming the right tail of order value specifically shrinks the suburban occasionwear baskets — the strategy's best evidence — which is why Section 4.5 insisted that legitimate extremes are kept and why “remove outliers” is never, by itself, an acceptable log entry. In each case the failure mode is not malice; it is a reasonable-looking default whose burden lands on a group nobody was watching, the same pattern Section 3.6.1 traced for proxy measures (Barocas & Selbst, 2016).

The discipline this guide asks for is therefore procedural, and it has three parts. First, no silent exclusions: every rule that removes or alters rows is written, counted, and carried into the analysis's methods note — “we excluded N order lines under the following three rules” is a sentence every downstream reader is owed. Second, examine what was removed: before certifying, profile the excluded rows as a group — which stores, channels, and customer types they came from — because exclusions that cluster are exclusions with a story, and the story belongs in the log whether or not it is convenient. Third, test the judgment calls that could matter: where a defensible alternative existed (fill missing discounts with 0 versus drop the

lines), the flag columns built in Section 4.11 make it cheap to check whether the headline conclusions survive the other choice, and a conclusion that flips on a cleaning judgment is a finding about fragility that decision-makers must hear. None of this slows the work much. All of it converts editorial power exercised silently into editorial power exercised accountably — which is the only kind an analyst of record should want.

> **CONCEPT**
>
> **The Documented Exclusion**
>
> Every rule that removes or alters rows during preparation must be written as a rule, counted in the verification log, profiled for whom it disproportionately affects, and disclosed wherever the resulting analysis is used. Cleaning exercised without this discipline is silent editorial power over what the organization gets to see; cleaning exercised with it is a defensible analytic decision.
>
> Source: Adapted from Rawson and Muñoz (2019) and Gitelman (2013).

4.15 Chapter Summary

This chapter took StyleCraft from a file that could not be trusted to a table of record, and built the disciplines the trip requires. The main point is that cleaning is the first analysis: every preparation decision — what counts as a duplicate, which extremes are errors, what an absence means, which labels are the same fact, which rows are out of scope — changes the numbers every later chapter will report, and the difference between tidying and professional preparation is that professional preparation makes those decisions explicitly, predicts their effects, and documents them.

The chapter's instrument for that discipline is the verification log, which applies Chapter 1's predict-then-verify to every cleaning step: predicted row counts and totals before, actuals after, every mismatch explained before the pipeline proceeds, and the whole sequence anchored to a known external total. The log ends in a threshold rather than a feeling: a file is certified for a stated use when every prediction has been confirmed or explained and no quantity that use depends on remains unresolved, which is why the lab's miniature closes as provisionally cleaned rather than certified. Around the log, the chapter treated the defect families in turn. Missing values demand a mechanism diagnosis before a remedy, because dropping, imputing, and flagging each carry a nameable bias. Outliers are candidates for investigation rather than removal, detected by IQR fences, z-scores, and domain limits, and classified into legitimate extremes to keep and errors to treat. Duplicates come in exact and near forms, the near form turning on a documented judgment about which columns define an event's identity — and a third form, created by an unvalidated join, that inflates revenue without leaving a trace. Type coercion and categorical standardization make values comparable, with an accounting for everything that fails to convert and for the types the repairs themselves change.

On the constructive side, derived variables and deliberate grain changes through grouped aggregation extend the table's usefulness, verified by the totals an aggregation must preserve and

watched for the quieter failure in which an unresolved value is summed into a confident zero. Exploratory data analysis then closes preparation and opens analysis: disciplined questioning of distributions — their skewness and modality — and anomaly-first reading of histograms, box plots, bar charts, and scatterplots. Throughout, AI assistants draft the code while the analyst audits the counts, because the characteristic failures of delegated cleaning — silent row drops, wrong join keys, unstated imputation defaults — are exactly the failures a log catches. The ethics section named the stakes: cleaning is silent editorial power, and the documented exclusion is what makes it accountable.

Looking ahead, StyleCraft now has a certified table and a first look at its shape — including a right-skewed, arguably two-humped order-value distribution that hints at the company's two customer worlds. The next chapter puts the certified data to descriptive work: frequency tables, cross-tabulations, group comparisons, customer profiles, and the applied metric families of revenue, conversion, and retention — including the decomposition that finally explains what the suburban stores' high average order value is actually made of.

4.16 Exercises for Practice and Homework

The following exercises practice the chapter's main habits: diagnose the defect before treating it, predict the effect of every step, keep the log, and read exploratory output for anomalies. They are organized into three groups. Core chapter practice is the required path and should be completed by every student, and it holds the two homework submissions on which Assignment #1 draws directly. In-class activities are designed for discussion. Extensions are optional and go beyond the required material. Each exercise also carries its assignment label (required practice, homework submission, in-class discussion, or optional) so that instructors can assign selectively.

4.16.1 Core Chapter Practice

Exercise 4.1 Concept Check (Required Practice)

- State the four stages of the preparation pipeline and the cardinal sin of each.
- What four things does a verification-log entry record before a step is run, and why is a prediction-versus-actual mismatch described in this chapter as “the log working”?
- What does it mean to say a table is certified for a stated use, and what distinguishes a certified table from one that is merely cleaned?
- Explain the difference among MCAR, MAR, and MNAR using age_band in StyleCraft's customers table, and state which remedy-bias pairing each mechanism makes most dangerous.
- Why is “outlier” defined by distance rather than by error, and what evidence distinguishes a legitimate extreme from a genuine error in a transaction file?
- Why does an exact-duplicate check miss a re-exported order, and what judgment must the analyst make to catch it?

- A join to a dimension table can create duplicates that no unmatched-row count will reveal. Explain the mechanism, and name the check that prevents it.
- What does errors="coerce" do, and what verification must accompany any step that uses it?
- State two totals that must survive a line-to-order grain change exactly, and two reasons a groupby can silently fail to preserve them.
- In one sentence each: what does a histogram detect that describe does not, and what does a box plot show that a pair of group means hides?

Exercise 4.2 Diagnose the Defect (Required Practice)

For each symptom observed during inspection of a transaction extract, name the most likely defect family from Sections 4.4 through 4.7, the routine or check that would confirm the diagnosis, and the first remedy you would consider.

1. describe reports a maximum quantity of 999 while the 99th percentile is 6.
2. The sum of line_revenue is 4 percent above the finance system's figure for the same period.
3. value_counts on a category column returns both "Occasionwear" and "occasion wear."
4. A revenue column refuses to sum, and dtypes reports it as object.
5. info shows discount_pct with 96 percent non-null values while every other column is complete.
6. A join to the stores table leaves some rows with a missing opening_date.
7. A join to the products table leaves the row count 3 percent higher than it was before the join, and no row is unmatched.
8. A histogram of unit_price shows a lone bar far to the right of the catalog band.
9. Two rows share order_id, customer_id, product_id, quantity, and price but differ in order_line_id and by one day in order_date.

Exercise 4.3 Choose the Missing-Data Remedy (Required Practice)

For each situation, choose drop, impute (state which form), or flag — combinations are allowed — and state in one sentence the bias your choice introduces and why it is acceptable here.

1. 0.5 percent of rows are missing discount_pct; the data dictionary says the modal case is a full-price sale.
2. 12 percent of customers are missing age_band, and the missing share is three times higher among in-store shoppers.
3. campaign_id is null on 40 percent of order lines; the dictionary notes attribution is expected only for campaign-driven orders.
4. A handful of quantity values were set to missing after being classified as sentinel 999s, and the analysis that consumes the table is a revenue total.
5. A per-customer income field, self-reported and optional, is missing for a third of customers, and a colleague suggests filling it with the median.

Exercise 4.4 Predict the Row Count (Required Practice)

A raw extract has 50,000 rows, total revenue of $2,600,000, and inspection has found: 400 exactly duplicated rows; 250 rows forming 125 near-duplicate pairs on the identity columns; 60 rows with order_date before signup_date (exclusion rule agreed); and 900 rows missing discount_pct (to be filled with 0 and flagged). Write the verification log in advance: the predicted row count after each step in the chapter's default order, which steps change total revenue and in which direction, and which steps must leave it untouched. Then state the two questions you would ask before certifying if the actual final row count came back three rows lower than your prediction, and the one additional question you would ask if it came back three rows higher.

Exercise 4.5 Clean the Extract End-to-End (Homework Submission)

Complete Lab 4.1 Part B on the full transactions_raw file: copy the raw frame before the first transformation and keep it to the end, run the pipeline in a documented order, validate the cardinality of all three dimension joins, maintain the verification log throughout (baseline, one entry per step with prediction and actual, judgment entries flagged as such), pass the certification gate on unresolved revenue, reconcile the certified table against transactions_clean and against the known totals, and profile the excluded rows per Section 4.14 (which stores, channels, and customer types they came from). Submit the notebook, the log, and the completed AI-Use Appendix (Appendix D) recording every assistant exchange, including at least one case where the assistant's stated prediction and yours disagreed and how you resolved it.

Exercise 4.6 AI-Assisted Cleaning Audit (Homework Submission)

Give an AI assistant the miniature from Lab 4.1 Part A and this deliberately vague prompt, verbatim: “Clean this dataset and handle the missing values and outliers.” Run its code on a copy. Then audit: reconstruct from the output what the code actually did — rows dropped and under what implicit rule, values imputed and with what default, extremes treated and how, and whether it transformed the raw frame in place — and write a verification-log entry for each step the assistant took, marking every entry that records a decision the assistant made without surfacing it. Conclude with a rewritten prompt that would have forced the choices into the open, and submit the audit with the AI-Use Appendix.

4.16.2 In-Class Activities

Exercise 4.7 Find the Flaw in the Cleaning (In-Class Discussion)

Each scenario below describes a cleaning step that runs without error. Find the flaw, name the section whose discipline it violates, and propose the repair.

1. An analyst removes near-duplicates using the identity columns order_id and product_id only, and the row count drops by more than the near-duplicate inspection predicted.
2. To fix the currency strings, an analyst runs pd.to_numeric with errors=“coerce” and moves on; total revenue falls slightly.

3. An analyst deduplicates first and coerces unit_price afterward, and the O1009-style pair from Lab 4.1 survives into the certified file.
4. An analyst runs every cleaning step directly on the frame loaded from the raw file, then discovers at certification time that the near-duplicate rule was wrong and cannot show what the original labels were.
5. A left join to the products table runs cleanly, every row matches, and total revenue rises by 3 percent.
6. Asked to handle missing ages, an assistant fills them with the overall median; the suburban stores' average customer age falls by four years.
7. An analyst deletes every order above $300 as an outlier because “the catalog tops out at $90.”
8. A groupby to customer grain produces 7,912 rows, but the transactions file contains 8,000 distinct non-null customer_id values and some lines with customer_id missing.
9. An order-grain roll-up reports one order with revenue of exactly $0.00; the line-grain total reconciles perfectly.

Exercise 4.8 Ethics of Exclusion Mini-Cases (In-Class Discussion)

For each mini-case, identify who bears the burden of the cleaning decision, what the documented-exclusion discipline of Section 4.14 requires, and what you would write in the log.

1. Dropping all rows with missing age_band ahead of an age-based targeting analysis, given Exercise 4.3's pattern of in-store missingness.
2. Winsorizing order values at the 95th percentile “to stabilize the averages” in the expansion review.
3. Excluding the newest stores' first eight weeks of transactions as “ramp-up noise” from a store-performance comparison.
4. An exclusion rule for suspect dates that an analyst, short on time, applies only to the store channels because “that's where the problem was.”
5. A colleague proposes deleting the flag columns before circulating the certified file “to keep it tidy.”

4.16.3 Extensions

Exercise 4.9 Reflection Questions (Optional)

Finally, the following questions are intended to be thought-provoking rather than graded.

- This chapter claims that cleaning is the first analysis. What would have to be true for that claim to be wrong?
- Think of a number you have seen reported about an organization you know. What cleaning decisions must have stood behind it, and who would be able to tell you what they were?
- The chapter treats the raw file as evidence. What are the practical limits of that principle when the raw file itself is a vendor's export of something else?

- If an automated test suite can assert everything the verification log asserts, what is left for the log to do — and would you still keep one?
- Where should the line sit between a cleaning decision an analyst may make alone and one that requires a stakeholder's approval? Give an example on each side of your line.

4.17 Glossary of Terms

This glossary includes only the terms introduced in this chapter. Each definition is tied to the sources used in the chapter rather than added for decoration.

Data preparation. The disciplined transformation of raw source data into an analysis-ready table through a repeatable, documented sequence of inspection, cleaning, and verification, such that the result can be reproduced from the untouched raw file (adapted from Dasu & Johnson, 2003; Hellerstein, 2008).

Derived variable. A column computed from existing columns by an explicit, recorded rule at the grain of the table in which it appears — either directly from columns at the current grain or as a field of a coarser table produced by aggregation — documented in the data dictionary and verified against the totals it should preserve (adapted from Kimball & Ross, 2013; Wickham, 2014).

Domain limit. An allowed range or value set declared from business knowledge rather than inferred from the data — the catalog price band, the maximum permitted discount, a plausible basket size — used as an outlier-detection rule and recorded in the data dictionary's allowed-values field (adapted from Hellerstein, 2008).

Duplicate record. A row that records a real-world event already recorded by another row. Exact duplicates are identical in every column; near-duplicates match on the columns that define the event's identity but differ elsewhere, as when an order is re-exported with a new timestamp; and join-created duplicates arise when a merge to a dimension table finds more than one matching row per key.

Exploratory data analysis (EDA). The systematic examination of a dataset's distributions, groups, and relationships through summaries and simple charts, undertaken before formal modeling to discover structure, surface anomalies, and generate questions (adapted from Tukey, 1977; Behrens, 1997).

Flagging. The missing-value remedy that adds an indicator column marking which rows contained an absence, preserving the fact of missingness for later analysis. Flagging records missingness rather than resolving it and is normally combined with dropping or imputation.

Grouped aggregation. The operation, implemented in pandas by groupby, that collects rows sharing a key and summarizes each group, producing a table at a deliberately coarser grain; verified by the totals and counts the grain change must preserve.

Imputation. The replacement of missing values with estimated ones — a central value, the most frequent category, or a model-based prediction — preserving rows at the cost of manufactured data whose bias must be named (adapted from Little & Rubin, 2019; Osborne, 2013).

Missing-data mechanism. The process by which absences arise: missing completely at random (MCAR), missing at random given observed variables (MAR), or missing not at random (MNAR), where missingness depends on the unobserved value itself (Rubin, 1976; Little & Rubin, 2019).

Modality. The number of concentrations, or humps, in a distribution; bimodality often signals that a variable describes a mixture of distinct populations rather than one.

Outlier. An observation unusually far from the bulk of the data by a stated rule (interquartile-range fences, z-score threshold, or domain limits); a candidate for investigation and classification as legitimate extreme or genuine error, not for automatic removal (adapted from Barnett & Lewis, 1994; Tukey, 1977).

Sentinel value. A placeholder a source system records to signal a special condition — such as 999 for an unknown quantity — that masquerades as ordinary data; sentinels are classified as errors and typically converted to missing values with the mechanism documented (adapted from Hellerstein, 2008).

Skewness. The asymmetry of a distribution; right-skewed distributions concentrate values at the low end with a long tail of large values, the chronic shape of marketing quantities such as order value and customer spend.

Type coercion. The conversion of a column from its stored data type to the type its measurement requires, performed with an explicit accounting of every value that fails to convert and of any change the repair makes to what the column claims to measure (adapted from McKinney, 2022; Hellerstein, 2008).

Verification log. The running record, kept alongside cleaning code, in which the analyst predicts each step's effect on checkable quantities (rows, totals, missing counts) before running it, records the actuals after, and explains every mismatch; the chapter's instrument of predict-then-verify, and the basis on which a cleaned file is certified for a stated use once no quantity that use depends on remains unresolved.

Winsorizing. The treatment of extreme values by capping them at a chosen percentile rather than removing them; treated with caution in this guide because it alters real values wholesale and can shrink the decision-relevant tail of a distribution (adapted from Barnett & Lewis, 1994).

4.18 Further Readings

Students who want additional background may begin with the following readings. Cleaning-focused sources are listed first, exploration second.

- Dasu and Johnson (2003) for the classic practitioner treatment of exploratory data mining and cleaning, including the argument that preparation dominates the analytical workload.
- Hellerstein (2008) for a compact, systems-minded survey of quantitative data cleaning — outliers, deduplication, and integrity constraints — that maps closely onto this chapter's defect families.
- Little and Rubin (2019) for the authoritative treatment of missing data; the opening chapters cover the mechanisms at the depth this guide gestures toward.
- Wickham (2014) for the tidy-data principles that explain why some tables are easy to analyze and others fight back.
- Tukey (1977) for the founding text of exploratory data analysis, still bracing; Behrens (1997) for an accessible modern statement of the EDA attitude.
- Rawson and Muñoz (2019) for the argument against treating "cleaning" as neutral — the ethical companion to this chapter's log discipline.

4.19 References

Anaconda. (2020). *2020 state of data science*. https://www.anaconda.com/resources/whitepaper/state-of-data-science-2020

Barnett, V., & Lewis, T. (1994). *Outliers in statistical data* (3rd ed.). Wiley.

Barocas, S., & Selbst, A. D. (2016). Big data's disparate impact. *California Law Review, 104*(3), 671–732. https://doi.org/10.15779/Z38BG31

Behrens, J. T. (1997). Principles and procedures of exploratory data analysis. *Psychological Methods, 2*(2), 131–160. https://doi.org/10.1037/1082-989X.2.2.131

Dasu, T., & Johnson, T. (2003). *Exploratory data mining and data cleaning*. Wiley.

dbt Labs. (n.d.-a). *Data tests*. dbt Developer Hub. Retrieved July 12, 2026, from https://docs.getdbt.com/docs/build/data-tests

dbt Labs. (n.d.-b). *Severity*. dbt Developer Hub. Retrieved July 12, 2026, from https://docs.getdbt.com/reference/resource-configs/severity

Gitelman, L. (Ed.). (2013). *"Raw data" is an oxymoron*. MIT Press.

Hellerstein, J. M. (2008). *Quantitative data cleaning for large databases* (Technical report). United Nations Economic Commission for Europe. https://dsf.berkeley.edu/jmh/papers/cleaning-unece.pdf

Kimball, R., & Ross, M. (2013). *The data warehouse toolkit: The definitive guide to dimensional modeling* (3rd ed.). Wiley.

Little, R. J. A., & Rubin, D. B. (2019). *Statistical analysis with missing data* (3rd ed.). Wiley.

Matplotlib Development Team. (2026). *matplotlib.pyplot.boxplot*. Matplotlib documentation. Retrieved July 12, 2026, from https://matplotlib.org/stable/api/_as_gen/matplotlib.pyplot.boxplot.html

McKinney, W. (2022). *Python for data analysis: Data wrangling with pandas, NumPy, and Jupyter* (3rd ed.). O'Reilly Media.

Osborne, J. W. (2013). *Best practices in data cleaning: A complete guide to everything you need to do before and after collecting your data*. Sage.

pandas development team. (2026a). *Group by: split-apply-combine*. pandas documentation. Retrieved July 12, 2026, from https://pandas.pydata.org/pandas-docs/stable/user_guide/groupby.html

pandas development team. (2026b). *Nullable integer data type*. pandas documentation. Retrieved July 12, 2026, from https://pandas.pydata.org/pandas-docs/stable/user_guide/integer_na.html

pandas development team. (2026c). *pandas.Series.sum*. pandas documentation. Retrieved July 12, 2026, from https://pandas.pydata.org/pandas-docs/stable/reference/api/pandas.Series.sum.html

pandas development team. (2026d). *pandas.to_numeric*. pandas documentation. Retrieved July 12, 2026, from https://pandas.pydata.org/docs/reference/api/pandas.to_numeric.html

Rawson, K., & Muñoz, T. (2019). Against cleaning. In M. K. Gold & L. F. Klein (Eds.), *Debates in the digital humanities 2019* (pp. 279–292). University of Minnesota Press.

Rubin, D. B. (1976). Inference and missing data. *Biometrika, 63*(3), 581–592. https://doi.org/10.1093/biomet/63.3.581

Tukey, J. W. (1977). *Exploratory data analysis*. Addison-Wesley.

Wickham, H. (2014). Tidy data. *Journal of Statistical Software, 59*(10), 1–23. https://doi.org/10.18637/jss.v059.i10