

The machine learning pipeline

Hicham Janati

Fall 2026

Chapter 5 built predictors from a table it never questioned: every input already a number, on a sensible scale, with nothing missing and nothing wrong. This chapter is about building that table. A real estate agency hands us their brief (estimate the price of a listing from its characteristics) together with an export of 900 listings from their platform. The file contains everything a real file contains: a heavy-tailed price, typos, a column silently capped by the entry form, missing values, and a categorical variable no formula can multiply. Between this file and the clean matrix $\mathbb{X}$ of the previous chapters stands a sequence of decisions (what to drop, what to transform, what to impute, how to encode), and this sequence is not a preliminary to the learning: it is part of the learning, with parameters estimated from data, subject to the same discipline of training, validation and honest evaluation as the model itself. That composite object, transformations plus model trained as one and judged as one, is the *machine learning pipeline*.

Throughout, the course conventions hold: bold letters denote vectors and matrices, plain italic letters denote scalars.

1 Meeting the data

The agency's export has one row per listing and eight columns:

price (Dh)	surface	rooms	age	floor	elev.	dist.	district
438 000	74	2	12	3	1	2.1	Center
1 235 000	164	5	1	0	0	4.6	Marina
316 000	58	2	NA	1	0	3.4	Campus
4 180 000	87	3	34	5	1	6.8	Industrial
505 000	200	6	8	0	0	9.9	Suburbs

surface in m^2 , age in years, distance to the city center in km, elevator coded 0/1, and the district one of five: Center, Marina, Campus, Industrial, Suburbs. The target is the price. Two rows of the excerpt already hint at trouble (a missing age, and an 87 m^2 flat at 4.2 million dirhams), and finding such trouble systematically is the first task.

1.1 Look before you fit

The first step is descriptive, and it costs almost nothing: one histogram per column, before any model touches the file. Figure 1 shows them for the export, and every panel has something to say.

- *Price* is heavy-tailed (most listings sit below one million dirhams, a long tail follows), and a few isolated points float far beyond it, up to 20.5 million. The panel hides a further defect that the eye cannot catch: the *minimum* of the column is 0 Dh (three listings, saved before the price was typed, sitting invisibly inside the first bar). The summary statistics (min, max) complement the histogram.

- *Price per square meter*, a derived quantity worth a panel of its own, isolates the floating points: the bulk of the market sits around a median of 5 300 Dh/m², while five listings claim more than 20 000 Dh/m². Are they data errors? Not necessarily: a truly exceptional property could reach such figures, and the histogram alone cannot rule that out. What feeds the suspicion is the pattern: each of the five sits close to ten times a plausible value, the signature of a price entered with an extra digit. The pattern makes them suspect without proving an error; section 1.2 treats the two cases differently.
- *Surface* has a right tail and, at exactly 200 m², a spike of 25 listings (2.8% of the file). A spike at the top of the range comes from the entry form, not the market: the form caps the field at 200, and everything larger was recorded as 200. The column is *censored* at the cap.
- *Age* is bimodal (a burst of new builds, then a plateau of older stock), and 15% of its values are missing.
- *Distance* shows several bumps (the districts sit at different distances) and *rooms* is discrete: structure to remember, not problems to fix.
- Reading *across* the panels, finally: surface spans tens to hundreds of m², age runs from 0 to 80 years, distance from 1 to 13 km, elevator is a 0/1 flag. The columns live on entirely different scales: harmless for reading, consequential for fitting (section 2.2).

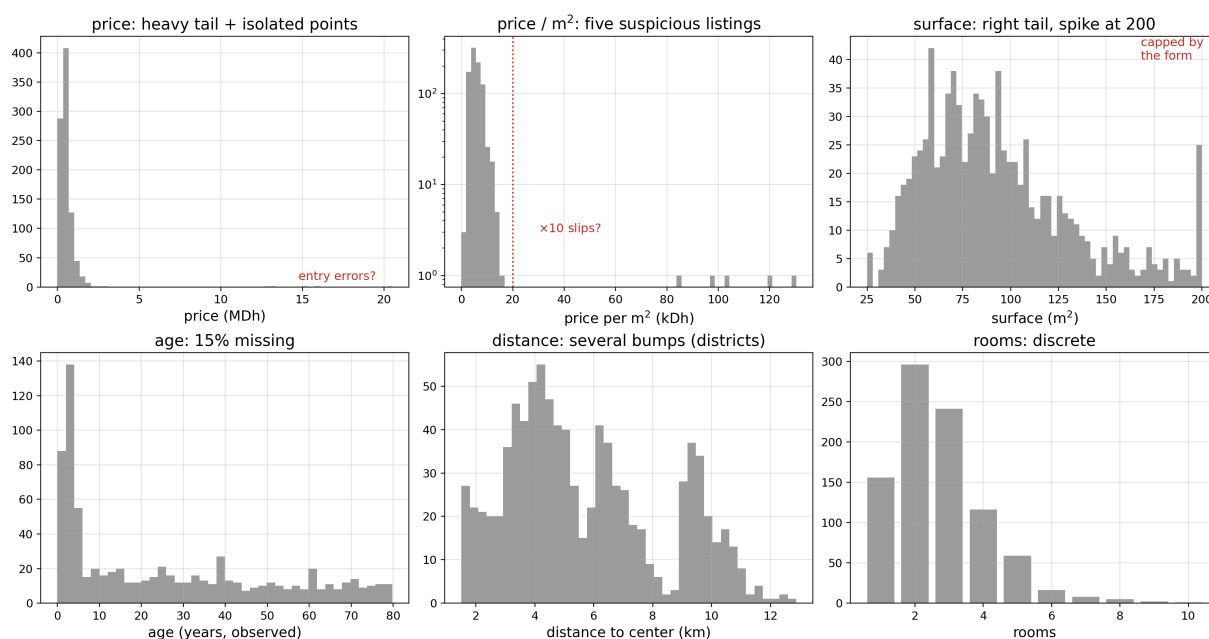


Figure 1: One histogram per column of the raw export. Price: heavy tail plus isolated points (and three zero prices hidden in the first bar). Price per m² (logarithmic count scale): five suspicious listings above 80 000 Dh/m², each near ten times a plausible value. Surface: right tail and a spike of 25 listings at the form’s 200 m² cap. Age: 15% missing. Distance and rooms: benign structure.

1.2 Dealing with what the histograms found

Each finding calls for a different treatment, and each has a cost if it is overlooked.

- *Impossible values* (the three zero prices) are errors beyond doubt: no listing costs nothing. Keeping errors has a measurable cost. A naive fit of the raw price on surface and district, trained on the dirty file, reaches a test error of 222 kDh per listing against 134 kDh for the

same model trained on the cleaned file, a 65% inflation contributed by eight rows out of six hundred. Since the target itself is false and unrecoverable, the rows are dropped.

- *Suspicious values* (the five listings above 80 000 Dh/m²) are very likely price errors. The right move is verification at the source: ask the agency, check the original advertisement. When verification is impossible, as here, the defensible choice is to exclude them *and record the decision* (five rows whose prices we could not certify) rather than silently keep or silently fix them. The cleaned file has 892 rows.
- *Censoring* (the surface cap) cannot be repaired: the information was destroyed at entry. Overlooking it has a targeted cost: the model never sees a surface above 200, so a 350 m² villa is priced as if it measured 200, a systematic underpricing of exactly the properties where errors are most expensive. The honest options are to keep the column knowing predictions above the cap are extrapolations (our choice), or to add an at-the-cap indicator so those rows can be treated separately.
- *Heavy tails* are genuine data, not errors, but they sit badly with squared-loss machinery: the loss weighs errors by their square, so the few most expensive listings dominate the training objective, and the fit follows them at the expense of the ordinary ones. The remedy is a transform, not a deletion (section 2.1).
- *Different scales* are not a defect of the file but a hazard for the methods: every tool that combines or compares columns in common units (a penalty on coefficient sizes, a distance adding squared differences) silently depends on arbitrary unit choices (sections 2.2 and 2.3).
- *Missing values* disable most of the machinery: a few learning algorithms tolerate holes natively, but the linear models of this course do not; a single missing entry breaks the matrix algebra. The choice is to drop the incomplete rows or to fill the holes by *imputation*, and it is measurable (section 2.5).

Every one of these findings came from a histogram or a summary statistic costing one line of code, and only the zero prices would have stopped a model (loudly, at the first logarithm). The dangerous defects are the silent ones: OLS on the raw file runs without a warning, with a twenty-million-dirham typo inside the squared loss.

2 Transforming the columns

2.1 The log transform

The price histogram is the same shape the store’s spend showed in chapter 4, and the same remedy applies: model $\log(\text{price})$ rather than price. Figure 2 shows the cleaned price before and after. On the raw scale the distribution is skewed and the mean (0.53 MDh) sits well above the median (0.44 MDh), dragged by the tail; on the log scale the histogram is near-symmetric and mean and median nearly agree (13.01 against 12.99). A symmetric, roughly Gaussian response is friendlier to squared-loss fitting (the loss is no longer dominated by the few most expensive listings), and the fitted effects become multiplicative, which is how housing markets are usually described (a percentage per district, not a fixed amount). The transform is a modeling decision like any other, and the chapter 4 caveat travels with it: exponentiating a prediction of the log returns a *median*-scale estimate in dirhams, not a mean.

2.2 Standardization

The remaining numerical columns live on arbitrary scales (square meters, years, kilometers, a 0/1 flag), and several tools of the course compare or combine them: penalties sum the sizes of

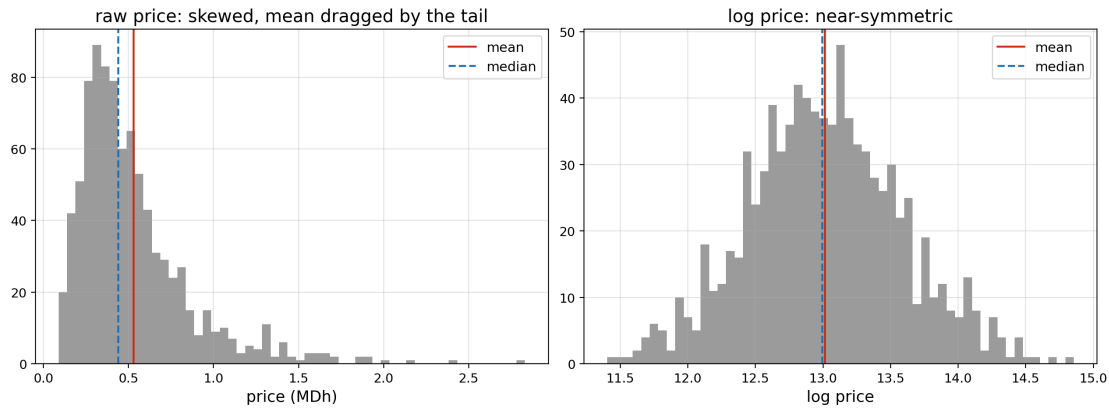


Figure 2: The cleaned price before and after the log transform. Raw: skewed, the mean dragged above the median by the tail. Log: near-symmetric, mean and median indistinguishable.

the coefficients, and distances (from chapter 2’s clustering onward) add squared differences across columns. *Standardization* replaces each column by its z-score,

$$z_{ij} = \frac{x_{ij} - \hat{\mu}_j}{\hat{\sigma}_j}, \quad \hat{\mu}_j, \hat{\sigma}_j \text{ estimated on the training rows}, \quad (1)$$

so that every column has mean 0 and standard deviation 1: the one-dimensional whitening of chapter 1, applied column by column. Two remarks matter. The operation has *parameters*: $\hat{\mu}_j$ and $\hat{\sigma}_j$ are estimated from data, a fact whose consequences section 3 develops. And the transformed column is unit-free: a coefficient on a standardized column reads “effect of one standard deviation”, comparable across columns.

The common alternative is *min-max scaling*, which maps each column onto $[0, 1]$ using its extremes:

$$m_{ij} = \frac{x_{ij} - \min_j}{\max_j - \min_j}, \quad \min_j, \max_j \text{ taken over the training rows}. \quad (2)$$

Both scalers are learned on the training rows and both remove the units; they differ in what anchors them. Min-max is the natural choice when an algorithm expects bounded inputs, and its output has a direct reading (0.7 means seventy percent of the way from the smallest to the largest training value). Its weakness is that the minimum and the maximum are the two most fragile numbers in a column: a single extreme value or surviving typo defines the whole range, crushing every ordinary value into a corner of $[0, 1]$; had the decimal slips of section 1 survived, the genuine prices would have occupied a tenth of the interval. Standardization’s mean and standard deviation are less fragile (though not immune, one more reason cleaning precedes scaling), and it is the course default, in particular before any penalty.

2.3 Standardization and penalization

Why chapter 5 insisted on standardizing before ridge and lasso can now be shown rather than asserted. The penalties act on the *size* of coefficients; but the size of a coefficient depends on the unit of its column (halve the unit and the coefficient doubles), so on raw scales the penalty compares quantities that are not comparable. Figure 3 fits the lasso to log price three times with the identical penalty $\lambda = 0.06$: once with surface in m^2 , once with surface in ares (one are = 100 m^2 , the same information in another unit), once on standardized columns. On raw scales the *selected feature set changes with the unit choice*: in m^2 the lasso keeps surface, age, distance and two district columns, and kills the elevator; in ares, rooms and elevator enter the kept set. The selection then reflects the unit choice rather than the data. The standardized fit keeps the six informative columns and wins on test error: RMSE 0.251 against 0.280 and 0.306 for the two

raw fits. Ridge is exposed to the same distortion for the same reason; standardization makes the penalty compare like with like, and chapter 5’s convention is justified.

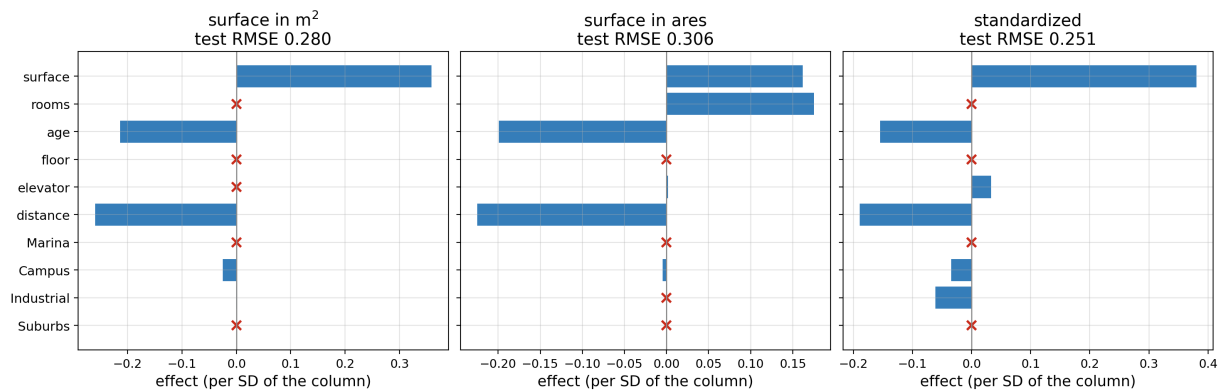


Figure 3: The lasso at the same λ under three encodings of the same file. Left and center: raw scales with surface in m^2 , then in ares; the kept set (blue) and killed set (red crosses) change with the unit choice. Right: standardized columns; the informative six survive, and the test RMSE is the best of the three.

2.4 Encoding categorical columns

The district column takes five values that no formula can multiply: “Marina” is not 2.7. The standard device is *one-hot encoding* of the input: replace the column by five binary columns (also called *dummy variables*), one per district, with a single 1 marking the listing’s district; figure 4 spells it out on five example rows. The device is the same one chapter 5 applied to the *labels* of a multiclass problem; here it acts on an input column, and the five binary columns then behave like any other covariates: each district gets its own additive effect on log price.

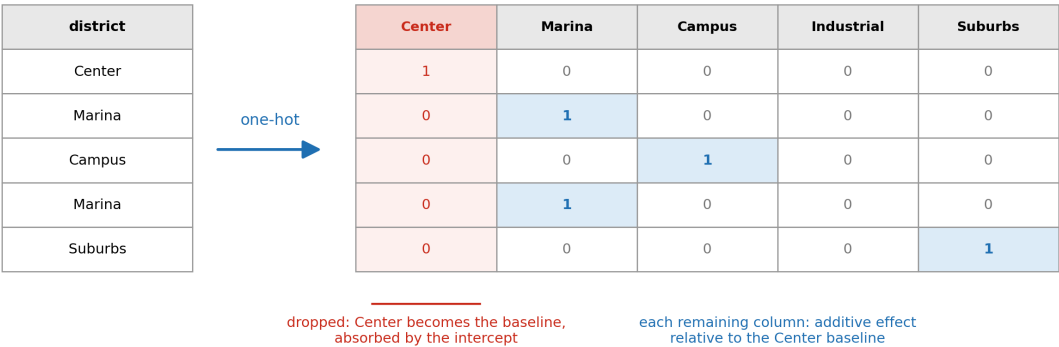


Figure 4: One-hot encoding of the district column on five example rows: each category becomes a 0/1 indicator column, with a single 1 per row. The first column (Center) is dropped: its rows are recognized by all indicators being zero, its price level is absorbed by the intercept, and the remaining coefficients read as effects relative to this baseline.

One algebraic trap comes with the encoding: the five columns sum to the constant column 1, which is exactly the intercept (perfect multicollinearity, chapter 4’s degenerate case). The standard remedies are to drop one category, whose effect is absorbed by the intercept and becomes the baseline the other coefficients are measured against, or to keep all five and let a ridge penalty make the optimization well-posed, as chapter 5 guarantees. We keep Center as the baseline.

2.5 Missing values

The age column is missing in 15% of rows, and the pattern is not random: cross-checking against the observed ages shows the gaps concentrate among older properties (old paperwork, undocumented buildings). Four standard strategies, fitted on the training rows and judged on the untouched test rows, give figure 5.

- *Drop the incomplete rows*: the default of much statistical software, and the worst of the four (test RMSE 0.211); it discards 91 of the 594 training rows, covariates and prices included, to avoid guessing one number.
- *Impute the training mean* of the observed ages (27.1 years): RMSE 0.204.
- *Impute the training median* (21.5 years): RMSE 0.209. Under this missingness pattern the median is the worse constant: the missing rows are mostly *old* properties, and the median, robust to the old tail, sits even further from them than the mean does.
- *Median plus a missing-indicator column*: RMSE 0.195, the best. The indicator lets the model learn that “age unknown” is itself informative; because the gaps are concentrated on old properties, the flag carries part of the signal the lost ages contained.

The general lessons: imputing beats deleting, because rows carry all their other columns; and when missingness is related to the hidden value (the common case in real files), a constant substitute plus an indicator recovers signal that the constant alone cannot.

Missing data is a field of statistics in its own right, whose central point is that *why* a value is missing matters as much as *that* it is missing. The standard taxonomy [1] distinguishes values missing *completely at random* (the gap is independent of everything, as with a server hiccup), missing *at random* (the gap is explained by other observed columns), and missing *not at random* (the gap depends on the hidden value itself, as in our file, where it is the old properties that hide their age). The defensible imputation depends on the mechanism: under the first, a constant substitute merely wastes signal; under the last, it is also biased, and devices like the indicator column (or model-based imputations beyond this course) become necessary. We keep the taxonomy as vocabulary and as a warning against treating imputation as a detail.

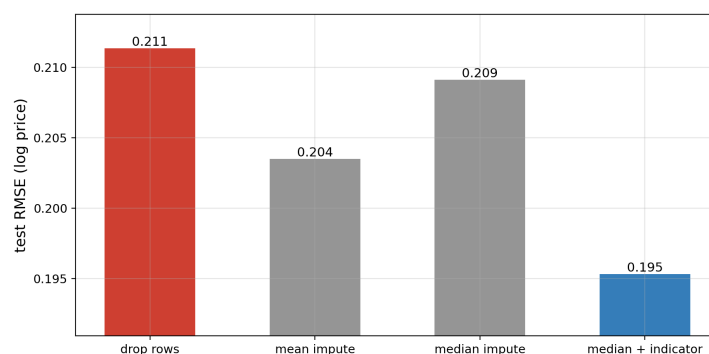


Figure 5: Four treatments of the missing ages, same model downstream, judged on the test rows. Dropping incomplete rows is the worst; a constant substitute plus a missing-indicator column is the best.

3 The pipeline and leakage

Assemble the chapter so far. Between the raw row $\mathbf{x}$ and the prediction stands a chain (screen the data errors, impute the missing age, encode the district, log the skewed columns, standardize),

and only then the model of chapter 5. Writing $\hat{\phi}$ for the chain and $\hat{f}$ for the fitted model, the deliverable is the composite

$$\hat{y} = \hat{f}(\hat{\phi}(\mathbf{x})), \quad (3)$$

and the hat on $\hat{\phi}$ matters: the chain has parameters (the screening threshold, the imputation median, the category list, every $\hat{\mu}_j$ and $\hat{\sigma}_j$), and all of them were *estimated from data*. One rule then follows from everything chapter 5 established about honest evaluation.

Key idea

Anything learned from data during preprocessing must be learned from the *training* data only: imputation constants, standardization means and variances, category lists, screening thresholds. A transformation fitted on rows that later serve as test rows has already shown the model part of the answers, and the test error it produces is no longer an estimate of anything. This contamination is called *leakage*.

Figure 6 draws the rule: parameters are estimated on the training rows, then applied (without refitting) wherever a row must pass through the pipeline, test rows included. Everything in this chapter respected the rule quietly: the medians, means and variances above were computed on the 600 training rows and merely applied to the test rows.

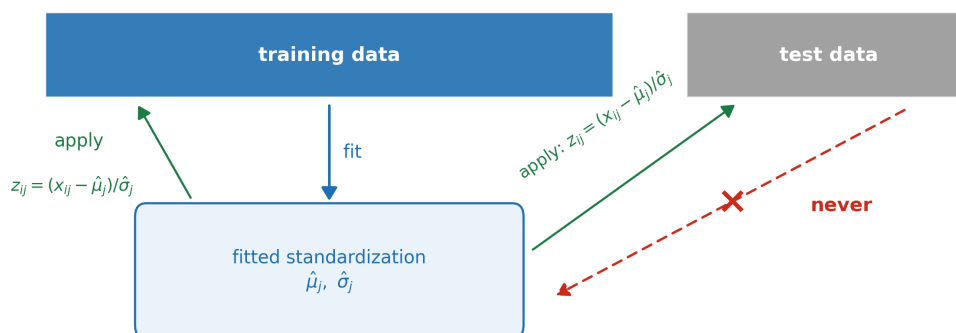


Figure 6: The fit/apply rule, shown on standardization alone: $\hat{\mu}_j$ and $\hat{\sigma}_j$ are estimated on the training rows only, then applied unchanged to any row entering the pipeline. The same rule holds for every fitted transform: imputation constants, category lists. Data never flows from the test rows into the fit.

The rule extends to validation: when cross-validation (chapter 5) evaluates the procedure, the unit being refitted on each fold's training part is the *entire* pipeline (3) (imputation and standardization recomputed inside every fold), not just the final model. Standardizing once on the full training set and then cross-validating only $\hat{f}$ leaks each fold's validation rows into the transforms.

Reality check

Leakage fails silently. A leaky pipeline runs without a warning, fits well, and reports validation scores that are honestly computed on contaminated inputs; nothing in the output betrays it. The only protections are procedural (one direction of data flow, from training rows to parameters to everything else) and the final, once-only test evaluation, which is also the only number a silent leak cannot inflate.

4 Evaluating predictors

The pipeline ends with a number: how good is the predictor? The loss that trained the model is not the only way (nor always the right way) to report its quality, and this section collects the standard metrics for each kind of target.

4.1 Regression metrics

On test rows $i = 1, \dots, n_{\text{te}}$ with predictions $\hat{y}_i$, the standard summaries are

$$\text{RMSE} = \sqrt{\frac{1}{n_{\text{te}}} \sum_i (y_i - \hat{y}_i)^2}, \quad \text{MAE} = \frac{1}{n_{\text{te}}} \sum_i |y_i - \hat{y}_i|, \quad R^2 = 1 - \frac{\sum_i (y_i - \hat{y}_i)^2}{\sum_i (y_i - \bar{y})^2}. \quad (4)$$

RMSE is the square root of chapter 5’s test error under squared loss, in the units of the target; MAE is the average absolute error, less sensitive to the few worst mistakes, and it is the metric matched to *median*-type predictions (the absolute loss is minimized by the median, as the squared loss is by the mean); R^2 compares the model to the constant predictor $\bar{y}$, and chapter 4’s caveats about it apply verbatim. On the housing test rows, the final pipeline of section 5 reaches RMSE 0.185 on log price (roughly a 20% typical relative error) and $R^2 = 0.90$; carried back to dirhams, the mean absolute error is 79 kDh and the median absolute error 58 kDh, against a median listing price of 431 kDh. Reporting the pair (RMSE, MAE) is good practice: a large gap between them is itself a diagnostic, flagging that a few large errors dominate.

4.2 Classification metrics

The agency’s second question is binary: which listings will sell within 30 days? Call them *hot*: the *positives*, against the not-hot *negatives*; they are 26% of the file. A penalized logistic regression on the pipeline’s features (price included, since an overpriced listing lingers) produces, for each test listing, a prediction: hot or not hot. Evaluating it means comparing predictions to truths while keeping the good and the bad predictions separate. Crossing the predicted label against the true one gives the *confusion matrix*:

	truth: hot	truth: not hot
predicted hot	TP = 44	FP = 16
predicted not hot	FN = 42	TN = 196

The naming reads T/F for whether the prediction was correct (True or False) and P/N for what was predicted (Positive or Negative). The correct predictions sit on the diagonal (the Trues: TP hot listings called hot, TN cold ones called cold), and the errors sit off it: FP, false alarms, and FN, misses. From these four counts, several metrics can be computed, each answering a different question.

- *Accuracy* = $(\text{TP} + \text{TN})/n_{\text{te}} = 0.81$: *how often are we right overall?* It averages over both classes, so the more numerous class dominates it.
- *Precision* = $\text{TP}/(\text{TP} + \text{FP}) = 0.73$: *when we flag a listing, how often is the flag right?* The metric of whoever acts on the flags.
- *Recall* = $\text{TP}/(\text{TP} + \text{FN}) = 0.51$: *of the truly hot listings, how many did we catch?* The metric of whoever pays for a miss: our classifier misses about half of the hot listings.

Precision and recall are reported as a pair, since either alone is trivially maximized: recall 1 by flagging everyone, precision by flagging almost no one.

4.3 The ROC curve

Where did the predictions of the confusion matrix come from? The logistic model does not output a class; it outputs a probability score $\hat{p}(\mathbf{x}) \in (0, 1)$, and the class is a decision taken from the score: predict hot when $\hat{p}(\mathbf{x}) > t$, with the *threshold* $t = \frac{1}{2}$ as the symmetric default (that is what the matrix above evaluated). The choice of t matters. Figure 7 shows the output probabilities of two scorers on the same eleven listings, true labels colored. Scorer A separates the classes around $\frac{1}{2}$ and the default threshold serves it well. Scorer B ranks the listings *perfectly* (every hot listing is scored above every cold one), but its scores sit low, so at $t = \frac{1}{2}$ it flags nothing and its recall is 0; at $t = 0.26$ the same scorer is flawless. A confusion matrix therefore confounds two different things: the quality of the scorer’s ranking, and the fit between the threshold and its scores.

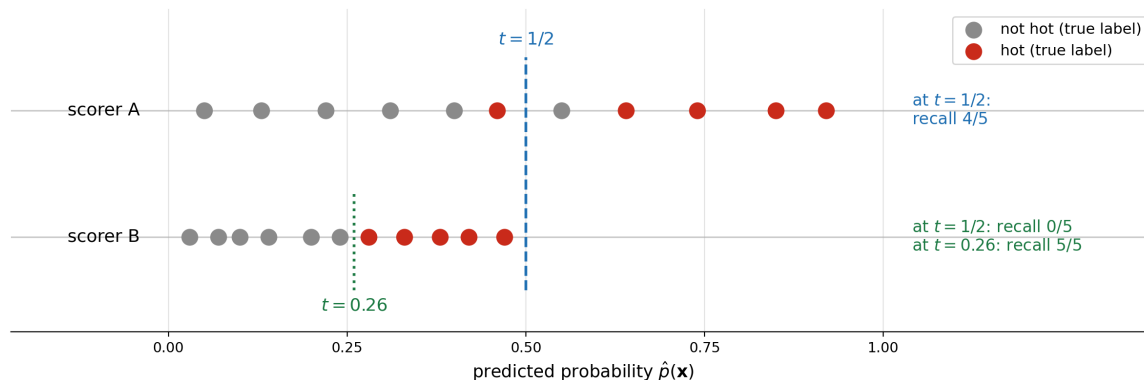


Figure 7: Two scorers on the same eleven listings (true labels colored). Scorer A works well at the default $t = \frac{1}{2}$. Scorer B ranks the listings perfectly, yet at $t = \frac{1}{2}$ its recall is 0, and at $t = 0.26$ it is 5/5. Measured performance depends on the threshold as much as on the scorer.

Comparing *models*, then, calls for a view that does not depend on any threshold at all; otherwise a competing model may look worse only because its threshold fits worse, exactly scorer B’s fate above. The idea is to *vary* t over its whole range and watch the model. Each t gives a (false positive rate, true positive rate) pair, and the curve these points draw as t sweeps from 1 to 0 is the *ROC curve* (figure 8), with random scoring on the diagonal and perfection in the top-left corner. The area under it, the *AUC*, summarizes ranking quality in one threshold-free number: it is the probability that a randomly chosen hot listing is scored above a randomly chosen cold one. The figure stages a model comparison: the full pipeline model ($AUC = 0.81$) against the same model deprived of the price columns ($AUC = 0.53$). The second curve hugs the diagonal: without the asking price, the model barely ranks better than chance, which identifies the price relative to the market as the signal that makes the classifier work. No choice of threshold can rescue the weaker model: its whole curve is dominated.

4.4 Metrics for imbalanced classes

The metrics above need more care when the two classes are far from balanced (a 1% fraud rate, a 0.1% disease prevalence). Start with accuracy. At a 1% fraud rate, the classifier “nobody is fraudulent” scores 99% accuracy while catching nothing at all. Even our mild imbalance distorts the reading: predicting “not hot” for every listing scores 0.71 on the test set, a strong-looking number for a model with nothing inside it. With unbalanced classes, accuracy is dominated by the easy majority; this is the *accuracy trap*, and it is why the class-specific pair (precision, recall), which exposes the empty classifier immediately (recall 0), matters more as the positives get rarer.

The ROC view has a subtler blind spot. Both of its coordinates are rates *within* one true class (the true positive rate divides by the positives, the false positive rate by the negatives), so the curve is unchanged however rare the positives are. When they are very rare, that invariance

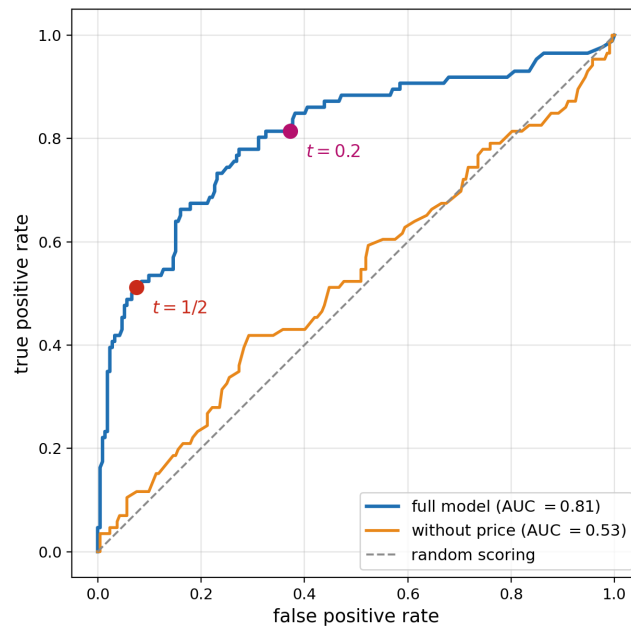


Figure 8: ROC curves of two hot-listing classifiers on test data: the full model (AUC = 0.81, with the operating points of $t = \frac{1}{2}$ and $t = 0.2$ marked) dominates the model without the price columns (AUC = 0.53), which hugs the random-scoring diagonal.

misleads. Screen 100 000 transactions of which 1 000 are fraudulent: a scorer that catches 800 frauds at a false positive rate of 2% has an excellent ROC point, yet 2% of 99 000 honest transactions is 1 980 false alarms, so the alarm list holds 2 780 names of which 71% are noise: precision 0.29. The ROC coordinates never see this, because the 1 980 false alarms are diluted by the huge negative class; the analyst who must work through the list sees nothing else.

The *precision–recall curve* repairs the blind spot: sweeping the threshold as before, plot precision against recall (figure 9). It differs from ROC in two ways. The random-scoring baseline is no longer a fixed diagonal but the horizontal line at the *prevalence* (flagging at random, the alarm list contains positives in their population proportion, 29% on these test rows), and the curve moves with the prevalence, which is exactly the sensitivity ROC lacks. The area under it (PR-AUC) is the imbalance-aware analogue of the AUC: 0.69 for the full model against 0.32 for the no-price model, the latter pinned near its baseline. The working rule: ROC–AUC measures the scorer’s intrinsic ranking quality; when positives are rare and the deliverable is a list someone must process, the precision–recall view is the one to report.

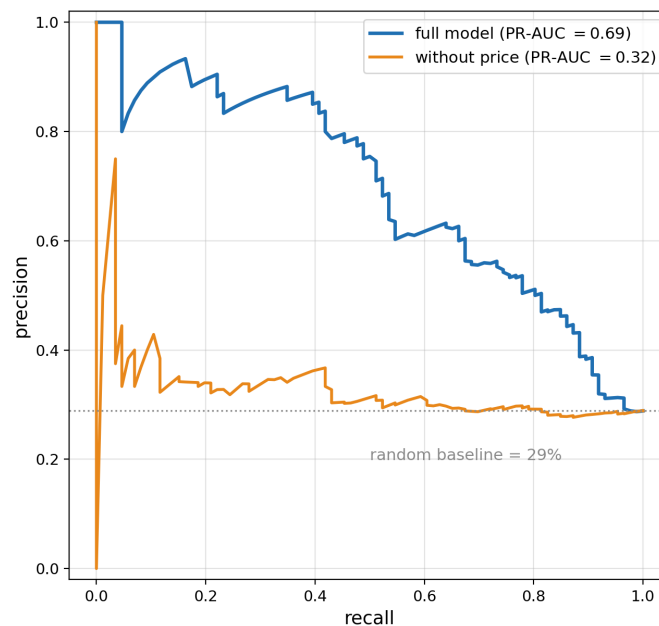


Figure 9: Precision–recall curves of the same two classifiers as figure 8. The dotted line is the random-scoring baseline, at the test prevalence; the no-price model barely leaves it (PR-AUC = 0.32 against 0.69 for the full model).

4.5 Calibration: are the probabilities real?

Logistic regression, like the softmax of chapter 5 and most classifiers met later, produces its scores through a sigmoid, so they land in $(0, 1)$ and are naturally read as probabilities. But nothing so far has checked that reading. When the model says “0.8”, do listings scored around 0.8 actually turn out hot about eight times in ten? Training minimized a loss; no step of it enforced that the numbers mean what they look like. A model whose scores do carry that meaning is called *calibrated*:

$$\mathbb{P}(Y = 1 \mid \hat{p}(\mathbf{X}) = p) = p \quad \text{for all } p \in (0, 1). \quad (5)$$

Checking it is an evaluation task in the spirit of this section: bin the test scores, and compare each bin’s mean predicted probability to its observed fraction of positives. Plotting the pairs gives the *reliability diagram* of figure 10: a calibrated model tracks the diagonal. Our cross-validated ridge logistic model does, roughly (test log loss 0.499). The second curve belongs to an *overfit* rival, a logistic model given all quadratic features and almost no penalty: among its test listings scored around 0.97, only 72% are actually hot, and among those scored near 0.02, 13% are; it is too extreme in *both* directions. Its ranking is not destroyed (its AUC is 0.74) but its probabilities are. Here overfitting has damaged the probability scale more than the ordering, and the log loss says so (0.858 against the ridge model’s 0.499).

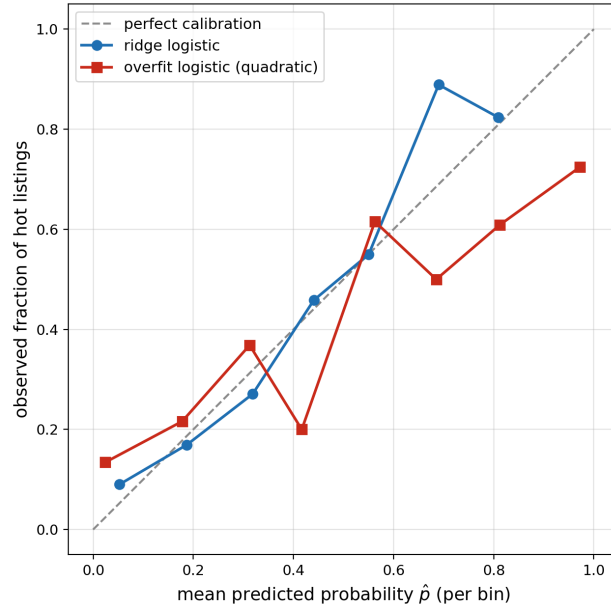


Figure 10: The reliability diagram: mean predicted probability per bin against the observed fraction of hot listings on the test set. The ridge logistic model tracks the diagonal; the overfit quadratic model is overconfident: fewer of its high-scored listings are hot than its scores claim.

A miscalibrated model can be repaired without refitting it. Recall from chapter 5 that the classifier’s probability is a sigmoid of an affine *score*: $\hat{p}(\mathbf{x}) = \sigma(F(\mathbf{x}))$ with $F(\mathbf{x}) = \beta^\top \mathbf{x}$. The idea is to leave the model alone and fit a one-dimensional correction map g on top of it, replacing $\hat{p}(\mathbf{x})$ by $p_{\text{cal}}(\mathbf{x}) = g(F(\mathbf{x}))$. Fitting g requires data of its own: on a *calibration set* (rows the model was not trained on, in the fit/apply discipline of section 3), run the fitted model and keep, for each of its K rows, the raw score and the true label:

$$(F_1, y_1), (F_2, y_2), \dots, (F_K, y_K), \quad F_k = \text{logit } \hat{p}(\mathbf{x}_k), \quad y_k \in \{0, 1\}.$$

These pairs are the training data *of the recalibration itself*. The scores play the role of a one-dimensional input, the labels are the targets, and the map is chosen by chapter 5’s own principle, minimizing the empirical log loss over a family $\mathcal{G}$ of candidate transformations:

$$\hat{g} = \arg \min_{g \in \mathcal{G}} \frac{1}{K} \sum_{k=1}^K -[y_k \log g(F_k) + (1 - y_k) \log(1 - g(F_k))]. \quad (6)$$

Two choices of $\mathcal{G}$ are standard.

- *Platt scaling* [2] takes the simplest family imaginable, a linear transformation of the raw score:

$$p_{\text{cal}}(\mathbf{x}) = \sigma(a + b F(\mathbf{x})), \quad F(\mathbf{x}) = \text{logit } \hat{p}(\mathbf{x}). \quad (7)$$

With this two-parameter $\mathcal{G}$, problem (6) is *exactly* a logistic regression of the labels on the score: one feature, two coefficients, solved by the same numerical machinery as chapter 5. The reading of the fit is direct: $b < 1$ shrinks an overconfident score toward the base rate, $b > 1$ expands a timid one, and a shifts the overall level.

- *Isotonic regression* [3]: $\mathcal{G}$ = all nondecreasing maps into $[0, 1]$, with no parametric shape at all, only the constraint that a higher score must not yield a lower probability. It is classically fitted with the squared loss: sorting the calibration scores $F_{(1)} \leq \dots \leq F_{(K)}$ and writing g_k for the value assigned at $F_{(k)}$,

$$\min_{g_1 \leq g_2 \leq \dots \leq g_K} \sum_{k=1}^K (y_{(k)} - g_k)^2, \quad (8)$$

whose solution is a monotone *step function* (computed exactly by the pool-adjacent-violators algorithm). More flexible than a line, it needs more calibration data to be trusted.

Figure 11 shows the Platt repair on the overfit model: the fitted map ($b = 0.27$, a strong shrinkage of the overblown scores) pulls the reliability curve back to the diagonal across the whole range of probabilities, and the test log loss falls from 0.858 to 0.550. Ranking is untouched: the map is monotone, so ROC curves and AUC do not move; only the meaning of the numbers changes.

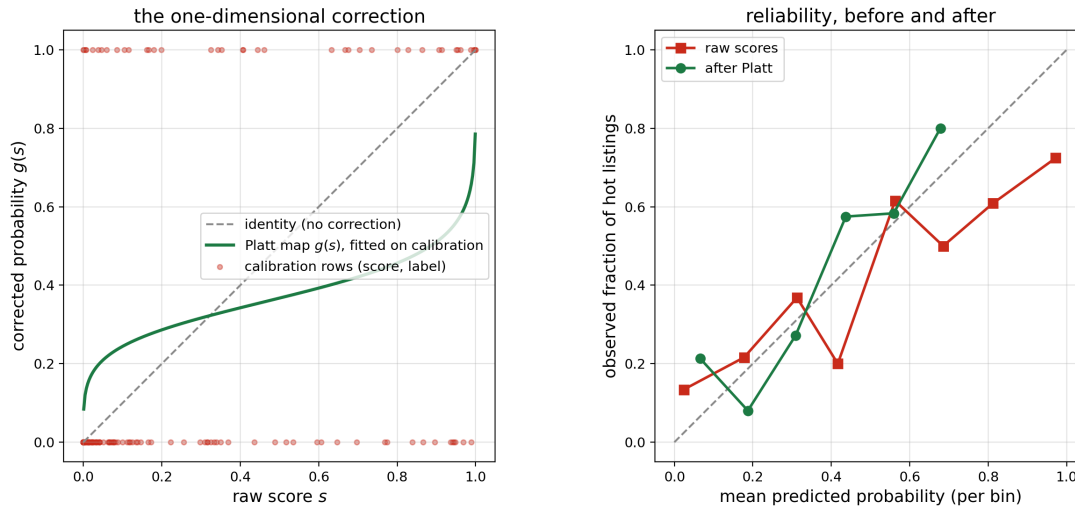


Figure 11: Recalibration. Left: the Platt map $p_{\text{cal}} = \sigma(a + bF)$, fitted on a left-out calibration set, shrinks the overblown scores. Right: applied to the test scores, it moves the reliability curve back to the diagonal across the full range; test log loss drops from 0.858 to 0.550.

Calibration matters exactly where probabilities are consumed as probabilities. A medical risk score drives treatment: “20% risk of complication” sits above or below a treatment threshold, and an overconfident model sends the wrong patients to surgery. A credit model’s default probabilities price loans and set the provisions a regulator audits; the numbers themselves, not the ranking, enter the arithmetic. Weather forecasting built its credibility on this property: “70% chance of rain” earned trust because, historically, it rains on about seven such days in ten. And any cost-based decision rule compares expected costs computed *from* the probabilities: with distorted probabilities it picks the wrong action even when the ranking is perfect.

5 Improving the model

Once the pipeline is fitted and evaluated, a second loop of work begins. The metrics say *how much* the model errs. The next question, *where* and *why* it errs, is what turns a mediocre pipeline into a good one.

5.1 Analyzing wrong predictions

The raw material of error analysis is the residuals, and one rule comes first: analyze the *training* (or validation) residuals, not the test ones; every look at test errors that leads to a change of the model turns the test set into training material, chapter 5’s rule in action. The method is to cross the residuals against everything: each covariate, each group, the predictions themselves, and to read the largest errors one by one.

Figure 12 applies this to the base housing pipeline. The right panel (residuals by district) shows nothing wrong: five boxplots centered on zero, no district visibly mistreated. The left panel tells another story: plotted against *surface*, the Marina residuals climb; the model underprices large Marina properties by +0.23 log units on average above 150 m², a 26% underestimate in

dirhams. The market prices sea-view space at a premium that grows with size; the model, forced to use one surface effect for all districts, splits the difference. The contrast between the two panels generalizes: errors that are *conditional* (wrong only for large surfaces in one district) are invisible in marginal summaries, and are found by crossing residuals against covariates, not by staring at one boxplot. For classification the same practice reads the confusion matrix cell by cell: pull the false negatives (hot listings the model missed) and false positives as two separate lists and look for what their members share.

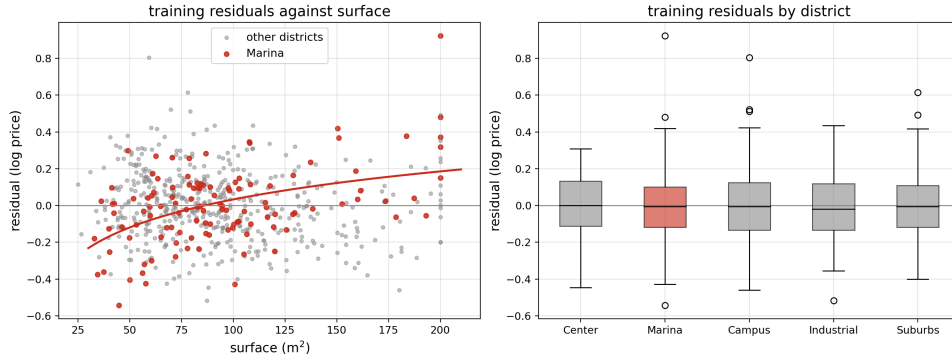


Figure 12: Error analysis of the base pipeline on training residuals. Left: residuals against surface, Marina highlighted; the rising trend is a missed size premium (+26% in price above 150 m²). Right: residuals by district, five unremarkable boxplots. A conditional error hides in marginal summaries.

5.2 Feature engineering

Error analysis found a pattern the feature set cannot express; the remedy is to extend the feature set. *Feature engineering* is the construction of new columns from existing ones so that a simple model can express what the analyst has understood. Here the finding dictates the features: an interaction column $\text{Marina} \times \log(\text{surface}/85)$ lets the surface effect differ in Marina, and a companion floor $\times$ elevator lets a high floor help with an elevator and hurt without one. Added to the same pipeline, refitted and revalidated identically, they cut the test RMSE from 0.195 to 0.185, a five percent reduction from two columns.

In symbols, the updated model appends the new columns to the old ones, each with its own coefficient:

$$\hat{f}(\mathbf{x}) = \hat{\beta}_0 + \hat{\beta}_{\text{surf}} \log x_{\text{surf}} + \cdots + \hat{\beta}_{\text{dist}} x_{\text{dist}} + \hat{\beta}_{\text{surf} \times \text{Mar}} \log x_{\text{surf}} \cdot x_{\text{Mar}}, \quad (9)$$

still a linear model in the sense of chapter 5’s remark; the new column is nonlinear in the raw inputs, the model stays linear in the coefficients, and nothing downstream changes.

The catalogue of constructions is small and reusable: interactions (a product of two columns, as above); ratios with domain meaning (rooms per square meter, for crowding); binned versions of a continuous column (age brackets) when an effect is not monotone; and the polynomial features of chapter 5 when curvature is suspected. Engineered columns are ordinary columns: they enter the pipeline before standardization, are subject to the same penalties, and are judged by the same validation. An engineered feature is a hypothesis about the data, and one that does not improve the cross-validated score is dropped. One warning goes back to section 3: a feature computed *from the target* (encoding a district by its average price, flagging errors found on test rows) is leakage in its most concentrated form, and it inflates the validation scores while failing in production.

The improvement loop is then: analyze the errors, engineer the features they suggest, revalidate, and stop when the validated score no longer improves.

5.3 Data drift and model monitoring

Every guarantee of this and the previous chapter (the test error as an unbiased estimate, the validated hyperparameters) rests on the i.i.d. thread running from the prologue: future rows resemble the rows the model learned from. In production that assumption decays. The market moves: prices inflate, a tramway line gentrifies Industrial, interest rates change what a fourth room is worth. The data channel moves too: the platform redesigns its entry form, the cap shifts from 200 to 300 m², a new field arrives half-empty. This slow divergence between the deployed model’s world and its training world is called *data drift*, and it comes in two flavors worth distinguishing: *covariate drift*, where the distribution of the inputs changes while the relationship to the target survives (more small flats get listed, but a small flat still costs what it did), and *concept drift*, where the relationship itself changes (the same flat no longer fetches the same price). Either way the failure is silent: the pipeline keeps producing predictions that have gone stale.

The response is *monitoring*: keeping the deployed model under the same statistical surveillance the training data received. In practice three things are watched. The *inputs*: the histograms of section 1, run continuously; the distribution of each incoming column is compared to its training counterpart, and a divergence (a new spike, a shifted mean, a surge of missing values) raises an alarm before any label is available. The *outputs*: the distribution of the model’s own predictions, whose drift is often the earliest symptom. And the *realized errors*: sale prices arrive with a delay, but once they do, the live error can be tracked against the test error certified at deployment, the one comparison that detects concept drift directly. The standard remedy is *retraining* on fresh data, on a schedule or when an alarm fires, often on a sliding window that lets the oldest rows expire; what is retrained is the entire pipeline of section 3 (transforms included) under the same train/validation/test discipline, with a fresh held-out period for the new certificate. With monitoring, a deployed pipeline is maintained over time rather than fitted once.

6 Conclusion

The pipeline replaces chapter 5’s clean-table assumption by a sequence of stages, each one learned, each one fitted on training data only:

stage	what is learned	fitted on
cleaning	screening rules (price/m ² cutoff)	training rows
imputation	constants (median age) + indicator	training rows
encoding	category list, baseline	training rows
transforms	log choices; $\hat{\mu}_j, \hat{\sigma}_j$	training rows
model	coefficients, hyperparameters	training rows (CV)
decision	threshold t^*	costs, validation
evaluation	RMSE / MAE / R^2 ; confusion, ROC, PR	test rows, once
monitoring	drift alarms, retraining triggers	production data

The pipeline is one predictor: everything between the raw row and the number is learned, so everything obeys the same train/validate/test discipline, and leakage is the silent violation of that discipline. And evaluation takes several metrics, not one number: RMSE against MAE, precision against recall. Choosing the metric means choosing which mistakes matter.

What the pipeline has not changed is the model at its end: linear in its features, every nonlinearity found by the analyst and installed by hand. The next chapter turns to predictors that discover such structure themselves.

References

- [1] D. B. Rubin. Inference and missing data. *Biometrika*, 63(3):581–592, 1976.

- [2] J. C. Platt. Probabilistic outputs for support vector machines and comparisons to regularized likelihood methods. In *Advances in Large Margin Classifiers*. MIT Press, 1999.
- [3] B. Zadrozny and C. Elkan. Transforming classifier scores into accurate multiclass probability estimates. In *Proceedings of the 8th ACM SIGKDD Conference on Knowledge Discovery and Data Mining*, 2002.