# Classification, Propensity, and Churn Models

*From How Much to Whether: Probabilities, Verdicts, and the Price of Being Wrong in Both Directions*

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


## Chapter Information

**ABSTRACT**

This chapter develops classification — the prediction of yes/no outcomes — as the completion of the machinery Chapter 8 built for numeric targets, with churn as its application. It constructs the churn label as a measurement decision, builds logistic regression from the log-odds equation, and adds classification trees and k-nearest neighbors as structurally different scorers compared on a training-only cross-validated leaderboard. It then builds the evaluation vocabulary: the confusion matrix in campaign terms, accuracy's failure under imbalance, precision and recall as trade-off instruments, ROC and AUC as threshold-free measures of ranking skill, and calibration as the separate property an economic threshold consumes. The destination is the decision threshold derived from asymmetric error costs, with propensity scoring, deciles, lift, and gains translating scores into budget-sized lists. The labs verify every figure by hand, keep the test set sealed until a finalist is chosen, and close with a group-level fairness screen.

**KEYWORDS**

classification; churn; logistic regression; confusion matrix; precision; recall; ROC/AUC; decision threshold; propensity; lift

**VERSION AND DATE**

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

**SUGGESTED CITATION**

Mendoza, J. (2026). Classification, propensity, and churn models. In *Applied business analytics for marketing decision-making: Business analytics and data visualization* (Chapter 9, Version 1.0) [Open educational resource]. CC BY 4.0.

of GitHub, Inc. Product names are used for identification only and do not imply endorsement. StyleCraft Collective is a fictional company created for instruction.

**COMPANION REPOSITORY**
Datasets, notebooks, and figure sources for this chapter are in the Applied Business Analytics companion repository on GitHub.

## Chapter Learning Objectives

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

- Identify marketing's binary prediction questions, distinguish classification from class-probability estimation, and explain at intuition level why linear regression is the wrong instrument for a binary target.
- Define a churn label completely for a non-contractual business — activity window, activity definition, eligibility, and the snapshot structure of Section 8.3 — and explain why the label is a measurement decision with program consequences, per the four-element discipline of Section 3.5.
- Explain the logistic function and log-odds at intuition level, fit and read a logistic regression, and translate its coefficients into plain-language statements about odds and probabilities without overclaiming causation.
- Describe how classification trees and k-nearest neighbors score customers, state each method's characteristic strengths and failure modes, and explain why k-NN requires the feature scaling of Section 6.7.
- Construct a confusion matrix from a scored list and a threshold, name its four cells in business vocabulary, and verify its arithmetic — counts, accuracy, precision, recall — by hand.
- Explain accuracy's failure under class imbalance, compute the majority-class baseline of Section 8.7 as the working floor, and state the imbalance-handling options at concept level.
- Compute and interpret precision, recall, and F1 in campaign terms, and choose which to lead with based on the decision's cost structure.
- Read ROC curves and AUC at working level as measures of ranking skill, state the full range AUC occupies and what it does and does not certify, and treat implausibly high AUC as the leakage signal of Section 8.4.
- Distinguish a model's ranking skill from its probability calibration, evaluate calibration out of fold on training data, and state the condition calibration imposes on any economically derived threshold.
- Select a model family and its settings using stratified cross-validation inside the training customers, per Section 8.9, and open the sealed test set once for the finalist.
- Derive a decision threshold from stated false-positive and false-negative costs using the expected-value framing of Section 8.10, explain why the software's default of 0.5 is an

inheritance rather than a decision, and reconcile an economic threshold with a budget constraint.

Build ranked-targeting exhibits — decile tables, lift charts, and gains charts — and translate them into a treatment recommendation and a budget argument.

Verify an AI-built or vendor-supplied classifier with the five-point audit of Section 8.11 plus the classification supplement — base rate and majority baseline, hand-checked confusion matrix at the economic threshold, threshold re-derived from stated costs, calibration evidence — documenting the audit per Appendix D.

Examine a classifier's error rates across customer groups, explain how classification models can cause the outcomes they predict, and state the fairness and feedback-loop obligations that attach to acting on classifications at scale.

Chapter 8 ended with a mercy it named on its way out. The Backstage deliverable predicted a dollar amount, and a dollar amount can be missed by a little or by a lot: the mean absolute error priced the typical miss, the leaderboard priced the model's margin over the sorted spreadsheet, and the whole apparatus of honest evaluation ran on the fact that a numeric error has a size. Most of marketing's predictive questions refuse the mercy. Will this customer churn; will she respond to the offer; will he click, convert, come back — yes or no, where the decision the model feeds is not close or far but right or wrong, and where being wrong has two faces with two different prices. This chapter builds the machinery for those questions: the logistic model that turns features into probabilities, the trees and neighbors that offer second opinions, the confusion matrix that gives the two kinds of error their business names, the curves that grade a model's ordering skill, the calibration check that grades its probabilities, and — the chapter's destination, and the payoff of a thread planted seven chapters ago — the decision threshold, chosen not by statistical convention but by the asymmetric costs of the two mistakes. Every discipline Chapter 8 installed rides along unchanged: the frame is still declared before fitting, the label is still a manufactured measurement, leakage is still the canonical error, model selection still happens inside the training data, and the test set is still touched once. What is new is that the number the model produces is a probability, the number the meeting wants is a verdict, and the space between them is where this chapter's analyst earns the seat at the table.

CONCEPT

**What This Chapter Is Really About**

Chapter 8's machinery graded predictions by how far they missed. This chapter's *verdicts* cannot miss by a distance — a customer predicted to churn either does or does not — and that small change of geometry rearranges everything downstream. One number can no longer summarize performance, because a binary verdict fails in two distinguishable ways, and the two failures bill different departments: the offer wasted on a customer who was staying anyway is a marketing cost, and the customer lost because no one intervened is a revenue cost, and nothing in the mathematics makes them equal. The chapter's methods, however, do not produce verdicts. They produce probabilities, and a probability forecast can still be more or less wrong — predicting 0.90 for a customer who stays is a larger probabilistic error than predicting 0.55, and 0.51 and 0.99 yield the same verdict at a 0.5 cut while carrying very different confidence. Class verdicts therefore create two error types; probability forecasts additionally require evaluation of their ranking and their calibration, and this chapter grades all three.

The chapter's real subject is the sequence of separations this forces on the analyst: the score separated from the verdict, because the model produces a probability and someone must choose where to cut it; the model's ranking separated from its calibration, because a well-ordered list of scores can still be systematically too high or too low; the model's quality separated from the threshold's wisdom, because a well-ranked list can be cut in a foolish place; and the statistical output separated from the economic decision, because the threshold that converts scores into treatments is not a modeling parameter at all — it is a business policy, derivable from the costs of the two errors, and the analyst who cannot re-derive it from stated costs has delegated the most consequential number in the pipeline to a software default. Chapter 2 planted the idea that the cost of being wrong is asymmetric. This chapter is where that idea stops being a lens and becomes arithmetic.

Source: Course concept developed for this guide, informed by Provost and Fawcett (2013).

## 9.1 Marketing Decision Context: The Save Offer and the Default Threshold

Chapter 8's deliverable shipped, and the Backstage Preview list was only the top of the fall retention program. Its other end concerns the customers StyleCraft is losing. Chapter 5's certified findings put a number on the worry that opened this guide: repeat-purchase rates are weakest exactly where the expansion bet is placed, and a large fraction of customers who buy once never buy again. The program's answer is a save track — internally, the Comeback Edit: a personalized offer combining a $15-off-$75 voucher, free shipping, and a curated selection drawn from the customer's purchase history. Finance has costed the treatment at roughly twelve dollars per treated customer once redemption rates and margin give-up are blended. The question is the list. Who gets the Comeback Edit — and, harder, who decides who gets it, on what evidence, cut at what line? The save track goes live with the November journeys; the targeting rule must be locked in twenty-eight days.

Three answers are already circulating, and each conceals a failure this chapter exists to expose. The first is the incumbent rule, defended by the same CRM operations lead whose sorted spreadsheet fought honorably in Chapter 8: "Anyone with no purchase in ninety days is lapsing. Send the offer to the ninety-day-lapsed list. We have run win-back this way for two years." The second lives in the marketing platform StyleCraft already licenses, whose Churn Risk panel sits one click from activation behind the claim that ends meetings: "Our churn model is 89 percent accurate." Activate the flag, treat the flagged, done. The third comes from the growth lead, and it has the charm of arithmetic honesty: "Email is nearly free. Why are we rationing a retention offer at all? Send the Comeback Edit to everyone who has ever bought. You cannot mis-target a blanket." The CFO, attending because the offer is margin and margin is the CFO's, has a preemptive answer: the voucher is not an email. Every redemption is real money, a blanket send prices out at twelve dollars times the whole file, and the finance model says most of that spend would land on customers who were never leaving.

Each answer has a real virtue, and stating the flaws is this chapter's tour. The ninety-day rule is cheap, transparent, and — the labs will show — genuinely informative, because long recency is the strongest churn signal in StyleCraft's data by design. It is also a one-variable model wearing a policy costume, and its one variable is precisely the one Chapter 5's lumpiness findings booby-trapped: the Suburban Occasion cohort routinely goes ninety quiet days between legitimate seasonal baskets, so the rule sprays vouchers across customers mid-cadence while missing the discount-dependent buyer whose recent small purchase disguises an exhausted relationship. The platform's 89 percent invites the question Chapter 8 trained: accurate against what, measured how, at what threshold? This chapter adds a sharper one. If roughly seventy percent of eligible customers do not churn, then seventy percent accuracy is available for free by predicting that nobody churns, so 89 percent is a nineteen-point margin over the do-nothing rule — a margin that could be genuinely valuable or could be produced by the wrong mixture of errors, and the number cannot be interpreted at all without the base rate, the confusion matrix, the threshold, and the cost structure beside it. That is the chapter's real complaint about accuracy: not that 89 percent necessarily fails, but that accuracy alone cannot establish whether the model helps the retention program. And the platform's flag arrives as verdicts, not scores, which means someone else's threshold — almost certainly the software default of 0.5 — has already made StyleCraft's most consequential retention decision without attending a meeting. The blanket proposal fails on the CFO's arithmetic, but its deeper failure returns in Section 9.14: treating everyone is not a neutral act with wasted cost, because a retention offer reaches customers it can annoy, train to wait for discounts, or — the industry's most uncomfortable finding — remind to leave.

So the VP of Marketing has commissioned the analyst — still you — with a deliverable in five parts. A churn definition in writing, because "churn" is not a column StyleCraft possesses but a measurement the analyst must construct and defend. A scored model, selected inside the training customers and graded once out of sample against stated baselines including the ninety-day rule — but graded with a new vocabulary, because a yes/no decision fails in two directions

and one accuracy number hides the difference. A threshold derived from the program's own economics, shown as arithmetic the CFO can re-run on a napkin and accompanied by the calibration evidence that entitles the arithmetic to be believed. A ranked list with its lift and gains exhibits, because the budget conversation will not be "is the model good" but "how deep into the file do we fund." And a fairness appendix the VP did not ask for but will need, because the scored file includes the customers of the newest stores, and Section 9.15 will show that a naive churn model has already made up its mind about them for reasons that have nothing to do with their loyalty. The chapter builds those five parts in order: the target and the label, the scorers, the evaluation vocabulary, the threshold and the ranked-targeting exhibits, then the AI audit, the labs, and the three sections that rehearse the meeting.

### *9.1.1 Opening Case Questions*

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

- The ninety-day rule is both a candidate answer and, in Chapter 8's terms, a baseline with teeth. State what the rule would have to be beaten *at* — which metric, on which population — for the model to earn the budget line, and why "accuracy" is not an adequate answer.
- The platform's "89 percent accurate" claim needs several numbers beside it before it means anything. Name at least three, and name the section of this guide that equips you to demand each.
- The Comeback Edit costs about twelve dollars per treated customer. Describe the two ways a targeting rule can waste that money — one for each direction of error — and explain why the two wastes do not cost the same amount. Which earlier section of this guide predicted this asymmetry would become the central question?
- The blanket proposal cannot mis-target anyone, by construction. List two costs of treating a customer who was never going to leave, beyond the twelve dollars — and keep the list; Section 9.14's first vignette will add a third cost that surprises most students.

## 9.2 From How Much to Whether: Binary Targets and Why the Line Fails

Chapter 8 defined the predictive frame — unit, target, features, horizon — and nothing in that definition required the target to be a dollar amount. The frame machinery transfers whole; one sentence of reminder suffices, and the reader who wants the full apparatus has Section 8.2. What changes in this chapter is the target's type, in exactly Section 3.4's sense: spend_next6m was a ratio-level quantity, and the targets of this chapter are binary categoricals — two classes, conventionally coded 1 for the class the business is watching for (the positive class) and 0 for its complement. Marketing manufactures such targets constantly, because marketing actions are mostly binary at the customer grain: send or do not send, and the customer responds or does not; show the ad or do not, and she clicks or does not; the contract month arrives, and he renews or does not. Will she churn, will he respond, will they convert, will it bounce — the yes/no questions are not a special case of marketing prediction; they are its majority.

Two tasks hide inside "predict a yes/no outcome," and separating them now saves confusion for the rest of the chapter. Classification, strictly, assigns each case a predicted class: this customer will churn, that one will not. Class-probability estimation assigns each case a probability of the positive class: this customer's churn probability is 0.62, that one's is 0.08. The second task is the one this guide's methods actually perform and the one marketing decisions actually need, because a probability preserves the ordering and the uncertainty that a verdict throws away — and because, as Section 9.9 will show, the conversion from probability to verdict is a business decision that should not be made inside the model at all. The distinction sounds academic and is the chapter's load-bearing wall: nearly every misuse this chapter catalogs, from the platform's pre-thresholded flag to the assistant's celebrated accuracy, begins by collapsing the two tasks into one.

**DEFINITION**

**Classification and Class-Probability Estimation**

Classification is the prediction of a categorical target: assigning each case to one of a fixed set of classes, in this guide's scope two. Class-probability estimation is the prediction, for each case, of the probability of the positive class — a score between 0 and 1 that ranks cases by likelihood and defers the assignment of verdicts to a separately chosen threshold. The methods of this chapter estimate probabilities; classifications are produced from them by thresholding, and the threshold is a decision, not a model property.

Source: Adapted from Provost and Fawcett (2013).

Why not simply reuse Chapter 8's instrument? Linear regression will run on a 0/1 target without complaint — the software fits, the coefficients print, and the output even has a respectable name in older literature (the linear probability model). The failure is not that it refuses to work; it is that it works badly in ways an intuition-level argument makes visible. First, the line does not know the target's boundaries: fit spend-style regression to a 0/1 churn outcome and it will cheerfully predict churn values of 1.3 for the most lapsed customers and −0.2 for the most engaged, numbers that cannot be probabilities and cannot be read as anything else. Second, the line's constant slope is the wrong shape for how binary risk behaves: a straight line says that thirty additional days of recency change churn risk by the same amount whether the customer is at the engaged extreme, in the uncertain middle, or already almost certainly gone — but risk near the floor and ceiling has almost nowhere to move, and the real action lives in the middle. The curve that honest binary risk follows is flat near 0, steep in the middle, and flat again near 1 — an S-shape, and Section 9.4's entire contribution is a principled machine for producing it. Third, the errors of a binary target misbehave in ways that undermine the least-squares machinery of Section 7.6 — a technical point this guide leaves at one sentence, because the first two arguments already carry the verdict: the right instrument for a bounded, S-shaped quantity is a model built for bounded, S-shaped quantities.

One more piece of vocabulary completes the setup, and it is a number rather than a method: the base rate, the share of the positive class in the population — the fraction of StyleCraft's eligible customers who actually churn in the outcome window. The base rate is the single most important number in any classification project and the one most reliably missing from celebrated results. It is the context that makes every other metric legible: an accuracy of 89 percent means one thing when churners are half the file and something entirely different when they are a tenth of it; a model that finds churners at twice the base rate is doing real work, and the same model quoted without the base rate is doing unverifiable work. This guide's rule, enforced in every lab and exercise: no classification result is reported without its base rate in the same exhibit, and Section 9.6 will show what happens to the analyst who forgets — the majority-class baseline of Section 8.7, which sat politely at concept level through the numeric chapter, is about to start winning arguments.

## 9.3 Defining Churn: The Label as a Measurement Decision

Section 8.3 established that a predictive label is manufactured, not found: a snapshot date splits time into a feature window the model may know and an outcome window the label summarizes, eligibility rules decide who gets a labeled row, and the whole construction is a measurement act carrying Chapter 3's obligations. That machinery transfers to this chapter unchanged — same snapshot of December 31, 2025, same eighteen-month feature window opening July 1, 2024, same January-through-June 2026 outcome window, same two-condition eligibility rule — and none of it is re-derived here; Section 8.3 is the reference. What this section adds is the problem that makes churn the most definitionally treacherous label in marketing: for StyleCraft, churn does not exist until the analyst defines it, and every element of the definition changes who the model learns to find.

The reason is structural, and it divides the industry's retention problems into two families. In a contractual business — wireless plans, streaming subscriptions, gym memberships — churn is an event: the customer cancels, the relationship has a legal off switch, and the data records the date it was flipped. In a non-contractual business like StyleCraft's, there is no switch. No customer ever tells an apparel brand she has left; she simply does not come back, and "has not come back yet" and "is never coming back" look identical in the transaction log for months. Chapter 5 measured this at the descriptive grain — repeat rates and their windows were metric constructions, per the four-element discipline of Section 3.5 — and this chapter inherits the problem at the label grain: churn must be *defined into existence* as a threshold on observed silence, and the definition is a choice among defensible alternatives, not a discovery of a fact.

**DEFINITION**

### Churn and the Churn Label

In a non-contractual business, churn is the sustained cessation of a customer's purchasing relationship, operationalized rather than observed: a customer is labeled as churned when a defined activity — usually a completed purchase — fails to occur within a defined outcome window following the snapshot date. The churn label therefore requires every element of the label construction of Section 8.3 plus a definitional commitment specific to churn: what counts as activity, how long the silence must last to count as departure, and which customers are eligible to be labeled at all. Different defensible definitions produce different labels, different models, and different treated populations.

Source: Adapted from Neslin et al. (2006) and Provost and Fawcett (2013).

The StyleCraft definition, stated in the four-element form of Section 3.5 because a label is a metric with a modeling job: churned equals 1 for an eligible customer with zero completed purchase orders in the outcome window, January 1 through June 30, 2026. Numerator concept: customers with no outcome-window purchase order. Denominator: none — it is a per-customer flag, not a rate. Window: the six-month outcome window, inherited from the Section 8.3 snapshot structure. Filters and eligibility: the Chapter 8 rule in both of its conditions — the customer's signup_date falls on or before December 31, 2025, **and** the customer completed at least one purchase between July 1, 2024 and December 31, 2025 — so that every labeled customer has a feature-window history to be scored on. Both conditions are load-bearing, and the second is the one drafts drop. Recency, average order value, discount share, and store share are all undefined for a customer with no feature-window purchases, because each divides by a quantity that is zero; a population defined only by signup date would therefore carry rows whose features do not exist. Customers who signed up and never purchased remain a named activation population, counted and reported, outside this churn model entirely.

One implementation detail carries more weight than its length suggests, and the labs enforce it. It is tempting to build the churn flag as the exact complement of Chapter 8's spend label — churned when spend_next6m equals zero — because the two are usually the same customers and the code is one line shorter. They are not guaranteed to be the same customers. Zero net revenue is not the same event as zero purchase activity: an order can net to zero after a discount and a returned line, purchases can be offset by returns and adjustments, and a customer can generate negative net revenue while having unambiguously transacted. The definition above says *activity*, so the label is built from counted purchase orders in the outcome window, and the reconciliation against spend_next6m is reported as a separate diagnostic rather than assumed. Where the two disagree, the disagreement is a finding about the revenue column, not a bug in the label — and the deliverable should not claim the churn flag is the exact complement of the spend label unless the dataset specification guarantees that every completed purchase produces strictly positive net revenue.

Every element of the definition is a lever, and moving any one of them changes the program. Table 9.1 walks the three that matter most. Shorten the outcome window to ninety days and the label sweeps in every Suburban Occasion customer sitting mid-cadence between seasonal baskets — the base rate inflates, the model learns to find slow shoppers rather than departing ones, and the Comeback Edit ships vouchers to customers who were three weeks from a full-price occasionwear order. Lengthen it to a year and the label grows more truthful about permanent departure but the program loses its purpose, because a customer silent for twelve months is far harder to save than one silent for five — the definition trades label purity against intervention timing, and the trade is a marketing judgment, not a statistical one. Redefine activity to include non-purchase engagement — an app open, an email click — and the label stops measuring commercial churn and starts measuring attention churn, a different construct that may or may not be what a voucher can treat. None of these alternatives is wrong. What is wrong is choosing one silently, and the deliverable's first page therefore states the definition, the rejected alternatives, and the reason — the same discipline the metric disputes of Chapter 3 taught, now with a model and a budget downstream of the choice.

Table 9.1

*The churn definition's levers and what each one changes*

| **Definitional lever** | **This guide's choice** | **What a different choice would change** |
|---|---|---|
| Outcome window length | Six months (the Section 8.3 outcome window) | Shorter inflates the base rate with mid-cadence occasion shoppers; longer purifies the label but treats customers too late to save |
| What counts as activity | At least one completed purchase order | Engagement-based definitions measure attention, not commerce; a net-revenue definition mislabels zero-value and fully returned orders as silence |
| Eligibility | Signed up on or before the snapshot **and** at least one feature-window purchase (Section 8.3) | Dropping the purchase condition admits never-activated signups whose ratio features are undefined, and changes both the base rate and the treated population |

One consequence deserves its own paragraph because it will resurface in three later sections. The definition makes the base rate a *constructed* number: StyleCraft's churn rate under this label is not a fact about StyleCraft so much as a fact about StyleCraft *under this definition*, and Chapter 5's certified repeat-purchase findings suggest it will land somewhere near a third of the eligible file — a designed expectation the labs will confirm against the shipped answer key. That

constructedness is why this chapter's evaluation sections insist on carrying the base rate everywhere: two churn models built on two defensible definitions can post identical accuracies while chasing different customers, and only the definition-plus-base-rate pair makes a reported result auditable. It is also why the platform's Churn Risk panel fails the first question before any statistics arrive: 89 percent accurate *at predicting what definition of churn* is not a rhetorical question, and a vendor who cannot answer it is selling a model of an undisclosed construct — the churn-flavored sibling of the undisclosed metric definitions that Chapter 3's one-metric-three-numbers dispute made famous.

> **CONCEPT**
>
> **The Label Is a Policy**
>
> A churn label does more than train a model; it decides, through the model, who is eligible to be saved. Customers outside the definition — the never-activated signup, the engaged browser who stopped buying, the customer who churned faster than the window can see — are invisible to the program built on it, not because anyone decided to ignore them but because the definition decided by default. This guide's standing rule follows: the label definition appears on the deliverable's first page, with its rejected alternatives and one sentence on who each alternative would have added or removed from the treated population. The rule costs a paragraph. It converts the definition from an assumption buried in code into a policy the meeting can see, discuss, and own — which is where a decision of this size belongs.
>
> Source: Course concept developed for this guide, informed by Neslin et al. (2006).

## 9.4 Logistic Regression: Probabilities from the Log-Odds Machine

The label exists; the first scorer can now be built, and this guide builds it from the instrument the reader already trusts. Section 9.2 established what the model must produce — an S-shaped probability, bounded between 0 and 1, steep in the uncertain middle and flat at the confident extremes — and logistic regression produces exactly that by the smallest possible modification of Chapter 7's machinery: keep the linear equation, change what it computes. A linear regression says the *outcome* is an intercept plus weighted features. A logistic regression says something one step removed: a quantity called the log-odds of the outcome is an intercept plus weighted features, and the probability is recovered from the log-odds by a fixed squashing curve — the logistic function — that maps any number the equation produces, however large or negative, into the interval the probability must occupy.

The intuition arrives fastest through odds, a currency marketing students already speak from every context where chances are quoted as ratios. A probability of 0.75 is odds of 3 to 1: three chances of the event for every one against. A probability of 0.5 is even odds, 1 to 1. A probability of 0.10 is odds of 1 to 9. Odds re-express probability on a scale that runs from zero to infinity instead of zero to one, and the *logarithm* of the odds runs the scale out symmetrically in both directions — negative log-odds for unlikely events, zero for even odds, positive for likely

ones. That unbounded, symmetric scale is what the linear equation lives on: features move the log-odds up and down freely, the way they moved dollars in Chapter 7, and the logistic function translates the result back into a bounded probability at the end. The S-shape falls out automatically: a one-unit feature change always moves the log-odds by the same amount, but the same log-odds movement changes the probability a great deal near 0.5 and almost not at all near the extremes — which is precisely the risk behavior Section 9.2 said a straight line could not express.

**DEFINITION**

**Logistic Function and Log-Odds**

The odds of an event are the ratio of its probability to its complement's: $p \div (1 - p)$. The log-odds are the natural logarithm of the odds — an unbounded, symmetric rescaling of probability on which zero means even odds. The logistic function is the inverse translation: it converts any log-odds value back into a probability between 0 and 1, producing the characteristic S-curve that is flat near both extremes and steepest at even odds.

Source: Adapted from Hosmer et al. (2013).

**DEFINITION**

**Logistic Regression**

Logistic regression models the log-odds of a binary target as a linear function of the features — an intercept plus weighted feature values, exactly the equation form of Chapter 7 — and converts the modeled log-odds into a predicted probability through the logistic function. Its coefficients are estimated from labeled data (by maximum likelihood, machinery this guide leaves to the software), and each coefficient is read as the change in log-odds — equivalently, the multiplicative change in odds — per one-unit difference in its feature, holding the model's other features constant.

Source: Adapted from Hosmer et al. (2013) and James et al. (2021).

Reading the coefficients is where Chapter 7's disciplines transfer with one translation step, and the translation is the only genuinely new skill. A logistic coefficient lives on the log-odds scale, which no meeting speaks; exponentiating it produces an odds ratio, which careful plain language can carry. A coefficient of 0.009 on recency_days exponentiates to about 1.009: each additional day since last purchase multiplies the odds of churn by 1.009 — about nine-tenths of a percent higher odds per day, compounding, so that ninety days of additional silence multiply churn odds by roughly 2.2. A negative coefficient on app_user exponentiating to 0.55 says app users carry a little over half the churn odds of comparable non-users. Three disciplines attach immediately, all inherited. The ceteris paribus boundary of Section 7.8 applies verbatim: each statement holds the model's other features constant and is silent on everything omitted. The verb registry of Section 7.2 applies verbatim: these are associations wearing precise multiplicative clothing, and "app users churn less" earns its verb while "the app retains customers" does not —

the customers chose the app, and Section 7.8's omitted-enthusiasm argument transfers without modification. And where categorical predictors do enter a model, the dummy machinery of Section 7.9 applies verbatim, with every odds ratio read *against the declared reference category* rather than absolutely. What does not transfer is any instinct to read effects on probability as constant: because of the S-curve, the same odds ratio moves a mid-risk customer's probability substantially and a low-risk customer's barely at all — statements about "percentage-point changes in churn probability" are only honest with a stated starting point, and the labs enforce the habit.

One design decision belongs here rather than in the lab, because it is a modeling judgment rather than a coding convenience. It is tempting to enter Chapter 6's named segments into this chapter's classifier as dummy variables, since the segments are memorable and the meeting speaks their names. This guide does not, for the reason Section 8.9 established: a segment assignment is a *learned* feature, produced by a scaler and a set of centroids estimated from data, so entering it as a precomputed column lets the customers being scored shape the definitions the model trains on — Table 8.2's split-contamination row, committed by a step nobody thinks of as a model. The core classifier therefore uses the behavioral variables the segments were built from, all of which the Chapter 8 feature builder already produces at the snapshot: recency, frequency, tenure, average order value, orders and revenue per month, discount share, store share, occasionwear share, and the two engagement flags. Segment membership returns in Section 9.12's fairness appendix as an external *profiling* variable — a lens for reading who the model is treating — which is the role it can play without contaminating anything.

One practical note belongs here before the reassurances. The coefficients this section teaches you to read are coefficients in *customer units* — days, orders, shares — and the predictive candidates of Section 9.12 will not supply them, because they standardize their inputs and carry the software's default regularization, which is right for prediction and wrong for reading. Code 9.6 therefore fits a small, separate model whose only job is interpretation, labeled in the lab as ineligible for the race. Keeping the reading model and the scoring model apart is the explanation-versus-prediction boundary of Section 8.2, drawn inside a single chapter.

One reassurance and one caution close the section. The reassurance: logistic regression is this chapter's primary instrument for the same reason linear regression anchored Chapter 7 — it is transparent, stable, fast, and its coefficients can be audited against known structure, which is what Section 9.11's audit will do with the dataset's designed churn drivers (long recency, single-purchase history, discount-only buying, and store-only suburban profiles all raise churn odds by design). The caution: everything the model produces is a probability estimate conditioned on the feature-window snapshot — a statement about customers who *resembled this one*, in exactly the sense Section 8.13 established for scores, and nothing in the sharpened vocabulary of odds changes what Section 9.13 will do to the sentence "the model says she is leaving."

## 9.5 Trees and Neighbors: Two More Ways to Score

Logistic regression reaches a probability through an equation; this section adds two scorers that reach one through entirely different structures, and the point of the addition is only partly the methods themselves. Chapter 8's comparison discipline — same folds, same metric, same baselines, one leaderboard fixed before the race — was built for exactly this moment: multiple candidate models, structurally dissimilar, competing for one budget line. The two candidates are chosen for their instructional contrast: one scores by asking questions, the other by finding resemblances, and each fails in a way that reviews a Chapter 8 lesson from a new angle.

A classification tree scores a customer by interrogation. The model is a sequence of yes/no questions about features — is recency_days greater than 120? is frequency equal to 1? is discount_share above 0.6? — arranged so that each answer routes the customer down a branch, until she lands in a leaf: a terminal group of training customers who answered every question the same way. Her predicted churn probability is simply the churn rate among the training customers in her leaf. The tree is grown from the data by recursive splitting: at each step, the algorithm chooses the question that best separates churners from non-churners among the customers reaching that point, then repeats within each branch (Breiman et al., 1984). Two properties make trees permanently useful in marketing rooms. They are readable — a modest tree prints as a flowchart a CRM manager can walk through, and its top question is a finding in itself (on StyleCraft's designed data, expect it to be a recency split, the ninety-day rule rediscovering itself as the first branch). And they are natively interactive in Section 7.9's sense — a tree that splits on discount_share only within the low-frequency branch has found, without being told, that the discount signal matters for some customers and not others. Their characteristic failure is the one Section 8.8 built the vocabulary for: an unconstrained tree keeps splitting until leaves hold a handful of customers each, memorizing training noise leaf by leaf — overfitting in its most visible form, with the telltale gap between training and cross-validated performance of Table 8.4, controlled in practice by limiting depth or leaf size. The lab manufactures exactly this failure on purpose, entirely inside the training customers, because a tree overfitting is the fastest visual demonstration of Chapter 8's U-curve a student will ever run.

**DEFINITION**

**Classification Tree**

A classification tree predicts by routing each case through a learned sequence of yes/no feature questions to a leaf, where the predicted class probability is the share of positive-class training cases in that leaf. The tree is grown by recursive splitting — repeatedly choosing the feature question that best separates the classes among cases reaching each point — and its flexibility is governed by limits on depth or leaf size, without which it overfits in exactly the sense of Section 8.8.

Source: Adapted from Breiman et al. (1984) and Provost and Fawcett (2013).

k-nearest neighbors dispenses with fitted structure altogether. To score a customer, find the k training customers most similar to her — nearest in feature space, by the distance machinery of

Section 6.6 — and predict the churn rate among those k neighbors. With k = 25, a customer whose twenty-five nearest training neighbors include ten churners is scored 0.40. The method's premise is pure resemblance: customers like you did what you will do (Cover & Hart, 1967). Its virtues are conceptual honesty — it is the "conditional average over similar customers" reading of Section 8.13 made literal — and freedom from any assumed equation shape. Its obligations and failures are equally instructive. Because it runs on distances, it inherits wholesale the scaling discipline of Section 6.7: unscaled, recency_days' hundreds dominate every 0/1 flag, and the neighbors are neighbors in recency only — the tyranny of units, third appearance, and the reason the lab's k-NN pipeline standardizes inside each training fold per the split-hygiene rule of Table 8.2. The choice of k is the bias–variance dial of Section 8.8 with a new handle: k = 1 memorizes (every training customer is her own nearest neighbor — training accuracy perfect, held-out accuracy poor), while enormous k averages everyone toward the base rate (the majority-class baseline rediscovered as a limiting case), and the lab therefore selects k by cross-validation inside the training data rather than by declaration. And k-NN produces no coefficients at all — nothing to read, nothing to audit against designed structure — which makes it this guide's standing example of the trade Table 7.4 named: predictive flexibility purchased with explanatory silence.

**DEFINITION**

**k-Nearest Neighbors (k-NN)**

k-nearest neighbors predicts a case's class probability as the positive-class share among the k training cases closest to it in feature space, under a stated distance measure computed on features scaled per Section 6.7. It fits no equation and learns no coefficients; its flexibility is governed by k, with small k overfitting and large k defaulting toward the base rate.

Source: Adapted from Cover and Hart (1967) and James et al. (2021).

Table 9.2 collects the comparison the leaderboard will run. Its last row states a designed expectation rather than a promise: on StyleCraft's data, whose churn structure was planted as a logistic relationship with a realistic noise ceiling, the three methods should land in a similar band with the logistic model competitive or ahead. That expectation is about *this dataset*, and it should not be generalized into a claim that method choice does not matter. The industry's own churn tournament found the opposite: across forty-five submissions on common validation data, differences in predictive accuracy between modeling approaches were large enough to change the profitability of a churn campaign by hundreds of thousands of dollars, with logit and tree approaches outperforming several alternatives (Neslin et al., 2006). The lesson the tournament actually teaches is the one this chapter's leaderboard enforces: approaches differ materially, so they must be compared on common validation data under a common campaign objective — which is a stronger argument for the leaderboard than the folk claim that any method will do.

Table 9.2

*Three scorers compared*

| Dimension | Logistic regression | Classification tree | k-NN |
|---|---|---|---|
| Scores by | Equation on log-odds | Learned yes/no questions | Resemblance to k nearest training cases |
| Output | Probability, smooth in features | Leaf churn rates (stepwise) | Neighborhood churn rate |
| Readable structure | Coefficients as odds ratios | The tree itself; top splits as findings | None — no structure to read |
| Needs scaling (§6.7) | Not for validity | No | Yes, inside each training fold (Table 8.2) |
| Setting chosen by cross-validation | None required at this level | Maximum depth | k |
| Characteristic failure | Misses non-additive structure unless told (§7.9) | Overfits via deep growth (§8.8) | Overfits at small k; scaling neglect |
| Designed expectation on StyleCraft | Competitive or ahead | Close behind, readable | Close behind with honest scaling |

## 9.6 The Confusion Matrix: Four Outcomes with Business Names

The scorers produce probabilities; suppose for the next three sections that a threshold has been chosen — the software's 0.5 will do for now, under protest that Section 9.9 will formalize — so that every customer carries a verdict. The instrument that grades verdicts is a two-by-two table so central to this chapter that its arithmetic is the chapter's verification theme: every count in it will be checked by hand in the labs, because it is the exhibit most often pasted into decks unexamined and most easily garbled in transit. Cross the truth against the prediction and every customer lands in exactly one of four cells. The model predicted churn and the customer churned: a true positive. Predicted churn and she stayed: a false positive. Predicted stay and he churned: a false negative. Predicted stay and she stayed: a true negative. The table of counts is the confusion matrix, and this guide's insistence — the reason Table 9.3 exists — is that the four cells are read in business vocabulary before any metric is computed from them, because each cell is a different event in the Comeback Edit program with a different line on the P&L.

**DEFINITION**

**Confusion Matrix**

A confusion matrix cross-tabulates actual against predicted classes at a stated threshold, partitioning all evaluated cases into true positives (predicted and actual positive), false positives (predicted positive, actually negative), false negatives (predicted negative, actually positive), and true negatives (predicted and actual negative). Every threshold-dependent classification metric — accuracy, precision, recall, and their relatives — is an arithmetic summary of these four counts, and changing the threshold changes the matrix.

Source: Adapted from Provost and Fawcett (2013) and Fawcett (2006).

Table 9.3 states the four cells in the program's own vocabulary, and its cost column is written to match the ledger Section 9.9 will derive rather than to sound expensive. Two of the four rows repay care, because the intuitive phrasing overcounts. A false negative does not cost the lost customer's entire future margin: the program could never have captured that margin with certainty, because only a fraction of treated would-churners are expected to be saved. What a false negative costs is the *missed expected incremental save* — the save rate times the retained customer's margin, which Section 9.9 will price at roughly forty dollars. A false positive does not cost the treatment plus margin given away, either: the twelve-dollar treatment cost is already a blended figure with redemption and margin give-up inside it, so adding margin a second time double-counts. What a false positive costs is the blended twelve dollars, spent on a customer who was staying.

Table 9.3

*The confusion matrix in Comeback Edit vocabulary*

| Cell | What happened | Program meaning | The cost, priced as Section 9.9 prices it |
|---|---|---|---|
| True positive | Offer sent; customer was leaving | A save opportunity reached | The $12 treatment cost, spent where the expected $40 save is available |
| False positive | Offer sent; customer was staying | A voucher sprayed at a loyal customer | The blended $12 treatment cost spent on a customer who would have stayed |
| False negative | No offer; customer left | A departing customer nobody tried to save | The missed expected incremental save value, estimated here at $40 — the expensive cell |

| Cell | What happened | Program meaning | The cost, priced as Section 9.9 prices it |
|---|---|---|---|
| True negative | No offer; customer stayed | The quiet majority, correctly left alone | Nothing — and the cell that inflates accuracy |

From the four counts, the first summary metric is the obvious one: accuracy, the share of all verdicts that were correct — true positives plus true negatives, over everything. Accuracy is legitimate arithmetic, this chapter's opening vocabulary note is hereby discharged (the word that was misapplied to a numeric target in Section 8.11's vendor deck belongs here, to verdicts), and accuracy is also the most reliably misleading number in applied classification, for a reason the fourth row of Table 9.3 telegraphs: the true-negative cell is usually enormous. Churn, response, conversion, fraud — marketing's positive classes are minorities, a situation called class imbalance, and under imbalance accuracy is dominated by the model's performance on the class nobody is asking about. The arithmetic is brutal and clarifying. If thirty percent of StyleCraft's eligible customers churn, the rule "predict that nobody churns" is seventy percent accurate while identifying not one customer to save. The platform's 89 percent begins to look different against that floor — not necessarily worse, but no longer self-explanatory — and the floor has a name the reader already knows: it is the majority-class baseline, defined in Section 8.7, held at concept level through the numeric chapter with a promise that it would do serious work here. This is the work. Every accuracy figure in this chapter's deliverables is reported beside the majority-class baseline and the base rate, and an accuracy whose margin over the do-nothing rule is never computed is Chapter 8's baseline discipline waiting to deliver its verdict.

**DEFINITION**

**Accuracy and Class Imbalance**

Accuracy is the share of evaluated cases whose predicted class matches the actual class. Class imbalance is the condition — usual in marketing — in which one class heavily outnumbers the other, and under it accuracy is dominated by the majority class: the majority-class baseline of Section 8.7 achieves accuracy equal to the majority share while identifying no positive cases at all. Accuracy is therefore reported only beside the base rate and the majority-class baseline, and never as a classifier's headline under imbalance.

Source: Adapted from Provost and Fawcett (2013) and He and Garcia (2009).

Imbalance also raises a modeling question students meet the moment they search beyond this guide: an internet's worth of advice recommends rebalancing the training data — resampling the minority class upward, the majority down, or synthesizing minority examples — so the model "sees" more churners. This guide's treatment is deliberately brief and at concept level. Rebalancing exists, sometimes helps ranking, and always distorts the model's probability scale (a model trained on a rebalanced world overestimates churn in the real one), which matters enormously in a chapter whose destination is a threshold denominated in real-world probabilities

and whose Section 9.8 makes calibration an explicit deliverable. The disciplined default for problems at StyleCraft's imbalance — a minority class near a third, not a thousandth — is to change nothing about the data and everything about the evaluation: grade with the base-rate-aware instruments this chapter builds, keep every split and every fold stratified so both sides of the wall carry the true base rate, and reserve rebalancing for the rare-event regimes where it earns its distortions. The one-sentence version students should carry: imbalance is usually an evaluation problem wearing a data-problem costume.

## 9.7 Precision, Recall, and F1 in Campaign Terms

Accuracy failed because it pooled the four cells into one number; the repair is to ask the two questions the program actually cares about, each a different ratio of the same counts. The first question belongs to the CFO, who is paying for the treated list: *of the customers we sent the offer, how many were actually leaving?* That is precision — true positives over all predicted positives, the purity of the treated list. Its complement is the waste rate: precision of 0.5 means half the Comeback Edit budget landed on customers who were staying anyway. The second question belongs to the CRM manager, who owns the churn number: *of the customers who were actually leaving, how many did we reach?* That is recall — true positives over all actual positives, the coverage of the at-risk population. Its complement is the leak: recall of 0.67 means a third of the departing customers walked out without anyone extending a hand.

**DEFINITION**

**Precision and Recall**

Precision is the share of predicted positives that are actually positive — of those treated, how many were the real thing; the purity of the target list. Recall is the share of actual positives that were predicted positive — of the real thing, how many were caught; the coverage of the at-risk class. The two are computed from the same confusion matrix, respond to the threshold in opposite directions, and answer different stakeholders' questions; neither is "the" performance number. Precision is undefined when nothing is treated, which is why a policy table that includes a do-nothing rule must handle the empty denominator explicitly rather than divide by zero.

Source: Adapted from Provost and Fawcett (2013) and Fawcett (2006).

The two metrics are structurally in tension, and the tension is the threshold made visible. Lower the threshold and the model calls churn more freely: the treated list grows, recall rises (fewer departing customers missed), and precision falls (more loyal customers swept in). Raise the threshold and the reverse. Neither direction is progress in itself — the movement just relocates cost from one cell of Table 9.3 to the other — and this is the precise sense in which Section 9.2 insisted the score be separated from the verdict: the model fixes how good the ordering is, and the threshold chooses where along the ordering the program stops treating, which is a question about the two cells' prices, not about the model. Students should therefore hear alarm bells at any unqualified claim that a classifier "has" a precision or a recall: it has one

*at a threshold*, and quoting either without the threshold — or quoting the pair achieved at different thresholds, a vendor-deck classic — is the metric-shuffling of Section 8.9 in its classification uniform.

When a single number is genuinely required — a leaderboard column, a tuning target — the convention is F1, the harmonic mean of precision and recall. The harmonic mean is the right kind of average for the job because it is dragged toward the smaller of the two: a model with precision 0.9 and recall 0.1 posts an F1 near 0.18, not the flattering 0.5 an arithmetic mean would print, so gaming one metric at the other's expense stops paying. F1's limit is stated with equal clarity: it weights the two errors equally, and this entire chapter exists because the two errors are *not* equally priced. F1 is a summary for contexts that have not yet priced their errors; Section 9.9 is for contexts that have, and StyleCraft, with twelve dollars against forty, is about to become one. Table 9.4 collects this section's vocabulary as working questions — the design rule the deliverable's evaluation exhibits follow, with each metric assigned to the stakeholder whose question it answers.

Table 9.4

*The working questions of classification evaluation*

| **Metric** | **The question it answers** | **Whose question it is** | **When it leads the exhibit** |
|---|---|---|---|
| Base rate | How common is churn under the stated definition? | Everyone; context for all else | Always present; never the headline |
| Accuracy | What share of all verdicts were right? | Nobody, under imbalance | Only beside base rate and majority baseline |
| Precision | Of those we would treat, how many are really leaving? | Finance — the budget's purity | When treatment is expensive or intrusive |
| Recall | Of those really leaving, how many would we reach? | CRM — the churn number's owner | When a missed positive is the dear error |
| F1 | One balanced number, both errors weighted equally | Leaderboards and tuning | When costs are unpriced — a temporary condition |
| Expected net value | What does this policy earn at the stated costs? | The meeting | Whenever the two errors have been priced |

## 9.8 Ranking Skill and Probability Quality: ROC, AUC, and Calibration

Precision and recall grade the model *after* a threshold has committed it to verdicts; this section grades the two properties of the scores themselves — how well they order customers, and whether the numbers mean what they say — before any threshold intervenes. The first instrument sweeps the threshold instead of choosing it. Start at a setting so high no one is called a churner, then lower it continuously; at each setting, record two rates: the true positive rate (recall — the share of actual churners correctly flagged) and the false positive rate (the share of actual non-churners incorrectly flagged). Plotted against each other across the full sweep, the pairs trace the ROC curve. A model with no ranking skill traces the diagonal — at every threshold it flags churners and loyalists in equal proportion, exactly what random guessing achieves. A model with perfect ranking hugs the top-left corner: all the churners flagged before any loyalist. Real models bow between the two, and the area under the curve — AUC — summarizes the bow in a single number, with an interpretation compact enough to survive a meeting: AUC is the probability that a randomly chosen actual churner receives a higher score than a randomly chosen actual non-churner, with tied scores contributing half credit (Fawcett, 2006).

**DEFINITION**

**ROC Curve and AUC**

The ROC curve plots a scorer's true positive rate against its false positive rate across all possible thresholds, tracing the trade-off between catching positives and falsely flagging negatives. AUC — the area under the ROC curve — summarizes threshold-free ranking skill and ranges from 0 to 1. An AUC of 0.5 indicates chance-level ordering; values above 0.5 indicate useful ordering in the stated direction; values below 0.5 indicate that the scoring direction is systematically reversed or worse than chance. AUC equals the probability that a randomly chosen positive case outranks a randomly chosen negative one, with ties counted as half. It is insensitive to the base rate and to any particular threshold: it grades the ordering, and neither the calibration nor the costs.

Source: Adapted from Fawcett (2006) and Google for Developers (2026).

Three properties define AUC's proper use in this guide, each earning a paragraph of discipline. First, AUC is a good general ranking diagnostic and a poor sole criterion. It cannot be flattered by imbalance and does not depend on a threshold nobody has justified yet, which is exactly why it is the right column for a first-pass comparison of structurally different scorers. But it grades ordering across the *entire* false-positive-rate range, and the Comeback Edit decision does not live across the entire range: it lives near a particular economic threshold, inside a budgeted upper portion of the file, using probability values to compute expected economics. A model with the highest overall AUC can be inferior in the top two deciles, inferior near the 0.30 cut, worse calibrated, and worth less money. The frame therefore declares a two-stage rule rather than a single number, and the two stages answer different questions. AUC, cross-validated and compared fold by fold against a stated baseline, is the **model-family selection metric**: it chooses

which kind of scorer orders customers best, and it is the right instrument for that job precisely because it is threshold-free. Expected net value at the derived threshold, computed from *calibrated* out-of-fold probabilities, is the **policy-validation gate**: the selected model must clear it, against zero and against the incumbent rule, before any money is committed. Calibration is the condition that makes the gate computable at all, and precision, recall, and treated-list size at the stated threshold are the policy diagnostics reported beside it.

The ordering of those two stages is not cosmetic. Ranking evidence is available from raw scores; economic evidence is not, because a dollar figure computed from uncalibrated probabilities is denominated in a currency the model does not print. So the family is chosen first on ordering, the chosen model is calibrated inside the training data, and only then does the arithmetic of Section 9.9 run. A leaderboard that carries a net-value column beside raw scores has quietly inverted that order, and the labs of Section 9.12 are built so that it cannot. This is Chapter 8's principle — grade the model on the artifact the meeting is choosing — carried into classification with one added clause: grade it in dollars only once the probabilities have earned the right to be multiplied by them.

Second, AUC has calibrated expectations, and the calibration is Chapter 8's too-good logic wearing this chapter's units. Individual churn behavior carries irreducible noise — the outcome window contains job changes, moves, weddings, and whims that no snapshot feature can see — and the StyleCraft data was designed with a realistic ceiling in mind, which the labs will document empirically for this population rather than assert from a general industry figure. That distinction matters: published churn AUCs vary substantially with the label, the horizon, the population, and the data available, so a number quoted as the industry's honest range certifies nothing about StyleCraft's. What transfers digit for digit is the reflex Table 8.4 installed. An AUC far above what the base's volatility, the honest baselines, and the label definition would support is not a triumph; it is a leak until proven otherwise — the label's ingredients have almost certainly contaminated the features, with the outcome-window recency of Table 8.2's third route as the usual suspect — and the auditor's first move is the feature trace, not the celebration. The labs manufacture this failure deliberately, as Chapter 8's did, because a leak a student has built is a leak she will recognize in a vendor deck.

Third, AUC is silent about the property the next section's arithmetic actually consumes. AUC grades ordering; the threshold derivation grades *levels*. A model can rank customers beautifully while producing probabilities that are systematically too high or too low — multiply every score by 0.6 and the ordering, and therefore the AUC, does not change at all, while every economic calculation built on those scores does. The property that closes the gap is calibration: a scorer is well calibrated when, among customers scored near 0.30, roughly thirty percent actually churn. Calibration is checked with a reliability exhibit — bin the scores, and compare each bin's mean predicted probability against its observed churn rate and its customer count — and, where the comparison is materially off, corrected with a cross-validated calibration procedure fitted inside the training data (scikit-learn developers, 2026a). The discipline that governs the correction is Chapter 8's: calibration is evaluated and, if necessary, repaired using training-only, out-of-fold

evidence, never by adjusting a model repeatedly against the test set until the reliability plot looks agreeable.

**DEFINITION**

**Calibration**

A probability scorer is calibrated when its predicted probabilities match observed event frequencies: among cases scored near p, about a share p are positive. Calibration is distinct from discrimination — the ranking quality AUC measures — and neither implies the other. It is assessed with a reliability exhibit comparing mean predicted probability against observed positive rate within score bins, alongside each bin's count, and it is the property that entitles a threshold derived from expected costs to be applied to a model's raw scores.

Source: Adapted from scikit-learn developers (2026a).

**CONCEPT**

**Ordering, Levels, and the Two Ways a Score Can Be Wrong**

A scored file can fail its program in two independent ways, and the deliverable reports on both. It can *order* customers badly, so that the funded portion of the list is not much richer in churners than a random portion — a failure AUC, lift, and gains detect. Or it can order customers well while getting the *levels* wrong, so that a score of 0.30 does not describe a thirty-percent risk — a failure only a calibration exhibit detects, and the one that silently corrupts every dollar figure downstream, because expected value is computed from probabilities rather than from ranks.

The practical consequence is a rule about which decisions each property licenses. Good ordering alone licenses rank-and-cut programs: treat the top N, fund the top two deciles, hand the CRM team a prioritized list. Good ordering *plus* calibration is what licenses the arithmetic of Section 9.9, where a probability is multiplied by a dollar benefit and compared against a dollar cost. An analyst who applies an economically derived threshold to uncalibrated scores has not made a small technical error; they have priced a decision in a currency the model does not actually print.

Source: Course concept developed for this guide, informed by scikit-learn developers (2026a) and Provost and Fawcett (2013).

## 9.9 The Threshold Decision: Asymmetric Costs, Operationalized

Section 2.7 introduced the asymmetric cost of being wrong as interpretive discipline: before any analysis, name which direction of error is dearer, because the answer should shape what the analysis treats carefully. Chapter 8 previewed the asymmetry twice — in the Backstage program's two error types and in the choice between MAE and RMSE — and promised that this chapter would make it the whole instrument. This section keeps the promise, and the registry entry deserves to be honored in the text as it was written: this is the operationalization of Section 2.7. The asymmetric-cost idea stops being a lens for reading results and becomes the arithmetic

that *produces* one — the decision threshold, the number that converts every score into a treatment and every error into a bill.

Begin with what the default conceals. Classification software, asked for verdicts, cuts at a probability of 0.5, and the choice feels so natural it is rarely recognized as one: call churn when churn is more likely than not (scikit-learn developers, 2026b). But 0.5 is the right threshold only under an assumption nobody at StyleCraft believes — that the two errors cost the same. Cutting at 0.5 says a wasted twelve-dollar voucher and a missed save are equivalent misfortunes, and that a customer with a 0.45 churn probability — nearly a coin flip on losing her entire future relationship — should be left alone because the coin leans slightly toward staying. The default is not a neutral setting; it is a cost assumption wearing a factory setting's camouflage, and the first act of threshold discipline is simply to see it. Two further conditions ride inside the equal-cost reading and should be stated with it: it assumes the scores are calibrated probabilities in the sense of Section 9.8, and it assumes the stated two-error ledger contains the relevant program economics — no additional benefit or cost attached to the true positives and true negatives, no class weighting, and no transformed probability scale quietly in play.

**DEFINITION**

**Decision Threshold**

A decision threshold is the probability above which a scored case is treated as positive — here, the churn probability above which a customer receives the retention treatment. The threshold is not a property of the model: it is a policy choice that allocates the model's inevitable errors between false positives and false negatives, and it is derived from the relative costs of the two errors, not from statistical convention. The software default of 0.5 embodies the assumption that the two errors cost the same, and inherits its authority from nothing else.

Source: Adapted from Provost and Fawcett (2013) and scikit-learn developers (2026b).

The derivation replaces the default with the program's own numbers, using the expected-value framing Section 8.10 introduced at concept level. Treating a customer costs money with certainty and produces benefit only probabilistically, so the treatment is worth extending exactly when the expected benefit clears the cost. Assemble the Comeback Edit's ledger. The cost side is finance's blended figure: twelve dollars per treated customer, redemption and margin give-up included. The benefit side requires two estimates that the analyst must obtain, document, and flag as estimates: the probability that a treated would-churner is actually saved — the save rate, which finance and CRM place at roughly one in four based on past win-back performance — and the value of a save, the retained customer's expected future margin, which finance carries at roughly $160 over the planning horizon. A treated true churner is therefore worth about 0.25 × $160 = $40 in expectation; a treated loyalist returns nothing the program can claim (she was staying) and costs the full twelve dollars. The treatment rule follows in one line: treat a customer whose churn probability is p when the expected benefit p × $40 exceeds the certain cost of $12 — that is, when p exceeds 12 ÷ 40 = 0.30. The threshold is 0.30, not 0.5, and the difference is not

a technical adjustment: at 0.5 the program treats only the customers the model is fairly sure about, while at 0.30 it deliberately accepts more wasted vouchers because the error the vouchers prevent is more than three times as expensive as the vouchers themselves. In other words — and this is the sentence the deliverable prints under the derivation — the threshold is where the program *chooses its mix of mistakes*, and the choice belongs to the people who pay for the mistakes, made once, in writing, in arithmetic a CFO can re-run on a napkin.

The derivation's honesty depends on four caveats, each a standing obligation. The first is calibration, and it is load-bearing rather than technical: the analytically derived threshold of 0.30 is valid only when the probabilities are sufficiently calibrated for the deployment population, because 12 ÷ 40 answers the question "at what genuine risk does the expected benefit clear the cost," and a score that is not a genuine risk cannot answer it. The deliverable therefore carries the training-only reliability exhibit beside the derivation, and where the exhibit shows material miscalibration, a training-only calibration procedure is applied before the threshold is used — never a correction chosen by watching the test set. The second is estimate uncertainty: the threshold inherits the uncertainty of its inputs, so the deliverable shows the sensitivity (at a $30 benefit the cut moves to 0.40; at $60, to 0.20), because a threshold that swings widely under plausible input changes is telling the program that its economics, not its statistics, are the binding uncertainty. The third is causal: the save rate smuggles in a claim that the offer *changes* some would-churners' behavior, which this chapter's observational machinery cannot certify, per the discipline of Section 7.2; the deliverable names the randomized holdout as the purchase that would certify it. The fourth is heterogeneity: the arithmetic assumed every customer shares one benefit figure, when Chapter 5's concentration findings guarantee they do not — a refinement (customer-specific benefit, hence customer-specific thresholds) the exercises explore and real programs eventually adopt.

> **BRIDGE**
>
> Chapter 11 supplies the instrument the third caveat names: a randomly held-out control inside the campaign, which is the only design that can turn the assumed save rate into a measured one.

One tension remains between this section and the budget, and resolving it is the deliverable's final piece of threshold work. The economic threshold answers "whom is it worth treating"; the budget answers "how many treatments we can afford," and nothing forces the two answers to agree. If the 0.30 cut marks 2,600 customers as worth treating and the November budget covers 1,600, the analyst faces a fork with two honest tines: cut by rank — treat the 1,600 highest scores, the capacity logic of Section 8.10's rank-and-cut, accepting that economically worthwhile treatments go unfunded — or take the gap to the meeting as a priced argument, because the lift machinery of the next section can state, in dollars, what the unfunded thousand treatments are expected to return. That argument — the model as a budget case, not just a list — is where this chapter's analyst does the most valuable work, and it is built from the exhibits the next section constructs.

CONCEPT

**The Threshold Is Where the Analyst Meets the CFO**

Every other number in the pipeline belongs to the analyst: the label, the split, the features, the leaderboard. The threshold belongs to the business, because it is nothing but a statement of relative prices — how many wasted treatments one missed save is worth — translated into a probability. The analyst's job is not to choose it; it is to *surface* it: to drag the cost assumption out of the software default, price the two errors with the people who own their budgets, derive the cut as arithmetic, show its sensitivity, evidence the calibration that entitles it to be applied, and record the decision.

An analyst who does this once changes how the room thinks about every scored program it funds afterward, because the question "what threshold, and from what costs?" — once heard — is impossible to unhear. This is the chapter's verification theme in its managerial form: any threshold you are handed, including your own software's, must be re-derivable from stated costs, and a threshold that cannot be is a cost assumption nobody agreed to.

Source: Course concept developed for this guide, informed by Provost and Fawcett (2013).

## 9.10 Propensity, Deciles, and the Lift and Gains Exhibits

The threshold converts scores to verdicts one customer at a time; this section builds the exhibits that convert the *whole scored file* into the meeting's language, and it starts by giving the score its industry name and disambiguating it. A behavioral propensity score is a model-estimated probability that a customer will exhibit a behavior — churn, respond, convert, upgrade — used to rank customers for differential treatment. The term is worth owning because it is the form in which this chapter's machinery is packaged everywhere in the marketing stack: every CDP "likelihood to buy," every CRM "response score," every retention flag is a propensity score with a wardrobe, and the questions this chapter has equipped — what label, what window, what leakage audit, what calibration, what threshold, against what baseline — are the questions that audit them all.

The disambiguation matters because the phrase has a second, established meaning that Chapter 11 will need. In causal inference, a propensity score is the estimated probability of receiving a *treatment* conditional on covariates, used to adjust comparisons between treated and untreated groups. This chapter uses *behavioral propensity score* throughout to mean a predicted probability of a marketing behavior. It is not the treatment-assignment propensity score of causal inference, and the two should never be substituted for one another in a sentence: one ranks customers for an offer, and the other tries to make an observational comparison behave more like an experiment.

**DEFINITION**

**Behavioral Propensity Score and Ranked Targeting**

A behavioral propensity score is a model-estimated probability of a defined customer behavior, used to rank customers for treatment. Ranked targeting treats customers in descending score order — to a threshold, a budget, or a capacity — so that the program's yield depends on the ordering's quality rather than on any individual score being exactly right. Its standard evaluation exhibits are the decile table, the lift chart, and the gains chart, all built by sorting the evaluated file by score and comparing capture against the base rate. The term is distinct from the treatment-assignment propensity score of causal inference, which estimates the probability of *receiving* a treatment and is introduced in Chapter 11.

Source: Adapted from Provost and Fawcett (2013) and Neslin et al. (2006).

The exhibits are built by one mechanical act — sort the evaluated customers by score, highest first, and slice the sorted file into ten equal deciles — followed by bookkeeping. For each decile, count the actual churners it contains and compute two quantities. Lift is the decile's churn concentration relative to the base rate: if the top decile's customers churn at 2.4 times the overall rate, the model's best ten percent is 2.4 times richer in the target class than a randomly chosen ten percent — a number with a built-in baseline, which is why it survives in industry vocabulary. Cumulative gains is the running capture: treating the top decile reaches what share of all churners; the top two deciles, what share; and so on to the whole file, where capture reaches one hundred percent by definition. Plotted, the gains curve rises steeply while the model's ordering is finding churners and flattens as the remaining file thins out, against the diagonal that random targeting would trace — the visual sibling of the ROC curve, redrawn in the units a budget meeting thinks in: how much of the problem do we reach for how much of the file.

The exhibits earn their central place because they price program designs directly, and the reader has seen this move before: it is the capture head-to-head of Section 8.10, matured into a full curve. If the top two deciles of StyleCraft's scored file capture on the order of half of all churners — the designed expectation for a model near the data's quality ceiling, which the lab will test — then the "treat the top two deciles" program reaches roughly half the at-risk margin for a fifth of the blanket program's cost, and the comparison against the alternatives is finally on one page in one currency. The ninety-day rule occupies a fixed point on the same axes (its treated list is one particular slice of the file, with its own capture), the blanket occupies the far corner (all cost, all capture), and the marginal logic of where to stop is read straight off the curve's slope: keep funding deciles while the incremental capture in a decile is worth more than the decile costs to treat — the threshold economics of Section 9.9 re-expressed as a stopping rule on a chart, which is how it most often survives contact with a budget meeting. The decile table, the lift chart, and the gains chart are, for ranked-treatment programs, what the leaderboard was for model choice in Chapter 8: the fixed exhibit family within which every candidate — model, rule, and blanket — competes on the same field, and this chapter's deliverable closes on them.

One reading discipline attaches to the decile table, because the draft version of this sentence is easy to overstate. Lift should generally be strongest in the early deciles and weaken as the list deepens; that is what a model with real ordering skill does. Perfect monotonic decline is not guaranteed in a finite evaluation sample, and a non-monotone middle is a finding about where the ordering blurs rather than an error to hunt down. What *would* be an error is a top decile whose lift sits near 1.0, because that says the model's best ten percent is no richer in churners than a random ten percent — the ranking has failed at exactly the end of the file the program funds.

## 9.11 AI as a Classification Assistant

The division of labor this guide has developed across three chapters — the assistant drafts mechanics, the analyst audits meaning — reaches its most consequential form here, because classification is where an assistant's unexamined defaults stop being statistical choices and start being business policy. A current assistant, handed the feature table and "build a churn model," will construct the label, fit three model families, tune them, print a confusion matrix, and report accuracy — and in that single fluent response it will have chosen a churn definition (undisclosed), a threshold (0.5, unmentioned), and a headline metric (the one this chapter spent Section 9.6 dethroning), while narrating the result with congratulations. None of these is a bug; each is a default standing in for a decision nobody was asked to make. The failure modes deserve their names, because the audit targets them.

The accuracy celebration: the assistant reports "91 percent accuracy" without the base rate or the majority-class baseline, and under imbalance the margin over the do-nothing rule is left uncomputed — the narration inherits the internet's enthusiasm for accuracy exactly as Chapter 8's assistants inherited its enthusiasm for $R^2$. The silent threshold: the confusion matrix arrives cut at 0.5, no costs discussed, and every downstream number — precision, recall, the treated-list size — quietly inherits the unexamined default; asked for "the model's precision," the assistant answers as if the threshold were a fact of nature. The label improvisation: asked for a churn model without a definition, the assistant invents one — often "no purchase in the last N days" computed *at the analysis date*, which builds the label out of the feature window and manufactures the outcome-window contamination of Table 8.2 in a single step; the resulting AUC is spectacular, and spectacularly meaningless. The test-set race: three models fitted and compared on test AUC, then a depth or a k chosen by watching the same test numbers move — the sealed set spent as a development instrument, which is Section 8.5's violation in its classification uniform. The rebalancing reflex: prompted with "imbalanced data," many assistants reach for resampling by default, distorting the probability scale that Section 9.9's threshold arithmetic consumes — a repair applied to a problem the evaluation should have absorbed. The uncalibrated economics: expected-value tables computed from raw scores with no reliability exhibit anywhere, so the dollars are denominated in probabilities nobody checked. And the verdict language: scores narrated as facts — "the model identified 412 customers who will churn" — the probability-to-verdict collapse of Section 9.2, performed by a sentence.

Where the assistant genuinely helps, use it deliberately, and the list is long because the mechanics are real work: the label-construction code from a written definition (the definition supplied, never delegated); the split, the stratified fold generator, the scalers inside the training folds, and the three-model scaffold from the declared frame; the confusion-matrix, reliability, decile, lift, and gains boilerplate, which is tedious and perfectly specifiable; error analysis — "which actual churners did the model score lowest, and what do they share?" surfaces the lumpy-cadence miss faster than any manual pass; and threshold sensitivity tables from stated cost ranges, which is exactly the well-specified arithmetic assistants execute reliably. The governing instrument extends Chapter 8's audit rather than replacing it: the five points of Table 8.5 run unchanged — frame, leakage, split hygiene, baseline, error in decision units — and Table 9.5 adds the four checks that classification makes necessary. Together they are this chapter's verification theme made procedural: the confusion matrix is re-added by hand, and the threshold is re-derived from stated costs, before any result is believed or repeated.

Table 9.5

*The classification supplement to the five-point audit (run with Table 8.5)*

| **Supplemental point** | **The check** | **Fails when** |
|---|---|---|
| S1. Base rate and floor | Base rate stated; majority-class baseline computed on the same evaluation set; accuracy read only against both, with the margin computed | Accuracy is quoted alone, or the do-nothing rule was never priced |
| S2. Matrix arithmetic | The confusion matrix's four counts re-added by hand: cells sum to the evaluation set; precision and recall recomputed from raw counts | Any count fails to reconcile, or metrics arrive without their matrix |
| S3. Threshold provenance | The threshold identified, its cost assumption stated, and the cut re-derived from the program's own FP/FN economics | The threshold is 0.5 by inheritance, or cannot be re-derived from any stated costs |
| S4. Calibration and selection hygiene | Model family and settings chosen by cross-validated ranking evidence inside the training customers; the selected model calibrated inside the training data; every expected-value figure computed from out-of-fold *calibrated* probabilities; the test set opened once | A net-value column sits beside raw scores; the calibration check covers a model other than the one selected; or a depth, a k, or a family was chosen by watching test performance |

**AI IN PRACTICE**

**The Classifier That Must Survive Its Audit**

The two-prompt pattern of Sections 7.11 and 8.11, with the supplement between. Prompt one, the frame as specification: paste the churn definition of Section 9.3 verbatim — snapshot, windows, activity rule, both eligibility conditions — plus the feature list with derivation rules and the snapshot-only instruction, the split protocol with seed and stratification, the declared selection protocol (stratified five-fold cross-validation inside the training customers), and the baselines named (a constant-score floor, a recency-only logistic baseline, and the ninety-day rule evaluated as a fixed policy). Then give the instruction that ends the improvisation: "Build exactly this pipeline. Select the model family and every setting using stratified five-fold cross-validation on the training customers only. Report one leaderboard graded on cross-validated AUC alone — no dollar figures beside raw scores. Then calibrate only the selected model, inside the training data, and report its out-of-fold reliability table and its cross-validated expected net value at the stated threshold. Do not touch the test set. Output scores, not verdicts. Choose no threshold yourself. Narrate nothing yet."

Then run Table 8.5's five points, followed by Table 9.5's four: confirm the label came back as defined and was built from purchase activity rather than net revenue; trace three features across the snapshot and run the recency range check of Section 8.12; confirm the scalers lived inside the folds and no test customer was consulted; read the cross-validated AUCs against the base's documented volatility, treating anything implausibly high as a leak to trace rather than a result to enjoy; check that the reliability exhibit describes the model that was actually selected and not some other candidate, and that every dollar figure in the reply descends from those calibrated probabilities; then — and only then — open the test set once, apply the threshold *you* derived in the Section 9.9 arithmetic, produce the confusion matrix, and re-add its four cells by hand against the test-set count. Prompt two, after survival: "Draft the recommendation paragraph for a marketing VP: the model's capture in the top two deciles against the ninety-day rule's, the treated-list size at the 0.30 threshold with the cost assumptions stated, and no sentence that asserts an individual customer will churn." Audit the draft's verbs per Section 7.2 — propensity language earns "is likely to," never "will" — file both exchanges per Appendix D, and sign the scored list as the analyst of record, threshold and all.

Source: Course concept developed for this guide, informed by scikit-learn developers (2026c) and Provost and Fawcett (2013).

## 9.12 Hands-On Application in Python and Google Colab

The preceding sections built the vocabulary; this section spends it on the Comeback Edit decision, in two labs that mirror the chapter's argument. Lab 9.1 makes every number touchable: the confusion matrix and the five-policy economics by hand at miniature scale, then the frame, the label, the base rate, and the sealed split at full scale, and finally an interpretive logistic model whose coefficients can be read in customer units. Lab 9.2 runs the race and carries it to the decision, in the order Section 9.8 declared: flexibility chosen inside the training folds by a one-standard-error rule, a five-candidate leaderboard graded on ranking alone, the finalist calibrated inside the training data and put through an economic gate, the sealed test set opened once for the

frozen model, the five competing policies priced on one table, the ROC, lift, and gains exhibits plotted, and the fairness appendix computed group by group.

The labs reuse Chapter 8's frame wholesale, and the reuse is made explicit in code rather than assumed: same snapshot of December 31, 2025, same feature builder from Codes 8.2 and 8.3, same outcome window, same two-condition eligibility rule, same seed. Two conventions hold throughout, and both are Chapter 8's disciplines made structural. Every model, including every baseline, is an estimator evaluated through the same cross-validation call on the same stratified folds, so that "same folds, same metric, same baselines" is enforced by the code rather than remembered by the analyst. And every step that learns anything from data — a standardizer, the estimator itself, a calibrator — lives inside a pipeline, so that it is refitted within each training fold and never estimates a quantity from a customer it is about to be graded on (scikit-learn developers, 2026c). AI assistants may draft any code cell (Appendix C has templates; Appendix A covers Colab mechanics); every output is predicted before it is computed, and every exchange is documented per Appendix D.

### *9.12.1 Lab 9.1, Part A: The Confusion Matrix and the Ledger in Miniature*

The miniature is the ten-customer table one last time, now carrying everything the running example has earned: the outcome-window activity from Lab 8.1 determines who actually churned (C002, C006, and C007), and a hypothetical logistic model, fitted to customers like these, has scored each customer's churn probability. The scores are chosen to be consistent with the designed churn drivers — long silence scores high, steady cadence scores low — and to contain the chapter's lessons in ten rows. Before running anything, predict which customers the model will get wrong at a threshold of 0.5, and in which direction.

**Code 9.1. Verify two confusion matrices by hand**

```
import pandas as pd

mini = pd.DataFrame({
    "customer_id": ["C001","C002","C003","C004","C005",
                    "C006","C007","C008","C009","C010"],
    "churn_prob": [0.05, 0.35, 0.15, 0.70, 0.10,
                   0.90, 0.60, 0.20, 0.65, 0.45],
    "churned":    [0, 1, 0, 0, 0, 1, 1, 0, 0, 0],
    # The incumbent's own column: no purchase in the 90 days
    # before the snapshot. A fixed policy, not a score.
    "lapsed_90d": [0, 0, 0, 1, 0, 1, 0, 0, 1, 1],
}).set_index("customer_id")

for t in (0.50, 0.30):
    pred = (mini["churn_prob"] >= t).astype(int)
    tp = int(((pred == 1) & (mini["churned"] == 1)).sum())
    fp = int(((pred == 1) & (mini["churned"] == 0)).sum())
    fn = int(((pred == 0) & (mini["churned"] == 1)).sum())
    tn = int(((pred == 0) & (mini["churned"] == 0)).sum())
    assert tp + fp + fn + tn == len(mini), "the matrix must sum"
    prec = tp / (tp + fp) if (tp + fp) else float("nan")
    rec  = tp / (tp + fn) if (tp + fn) else float("nan")
    f1   = 2 * prec * rec / (prec + rec) if (prec + rec) else float("nan")
    print(f"threshold {t:.2f}: TP {tp}  FP {fp}  FN {fn}  TN {tn} | "
          f"acc {(tp + tn) / len(mini):.2f}  prec {prec:.2f}  "
          f"rec {rec:.2f}  F1 {f1:.2f}")
```

*Expected output: two lines, one per threshold, each carrying the four cell counts and the four metrics computed from them; one assertion per threshold passes silently.*

Input: ten literal customers with a score, a truth, and the incumbent's flag. Transformation: two thresholds applied to one set of scores. Output: the two confusion matrices whose every figure the Verification Check re-derives by hand. The assertion is the smallest useful habit in the chapter: a matrix that does not sum to the evaluated population has lost customers somewhere, and a matrix that has lost customers is a matrix whose metrics mean nothing.

**VERIFICATION CHECK**

Before you run: build both matrices by hand, customer by customer. At the 0.50 threshold the model flags C004, C006, C007, and C009. True positives: C006 and C007. False positives: C004 and C009 — the two customers who ignite from nothing, whom the model (reading their long silence) reasonably condemned and who bought anyway. False negative: C002 — the occasion shopper whose big November baskets gave her a recent purchase and a comfortable 0.35 score, and who never came back: the lumpy-cadence miss, Chapter 5's finding wearing its third costume. True negatives: the remaining five. Accuracy is (2 + 5) ÷ 10 = 0.70; precision 2 ÷ 4 = 0.50; recall 2 ÷ 3 ≈ 0.67; F1 = 2(0.50)(0.67) ÷ (0.50 + 0.67) ≈ 0.57. At the 0.30 threshold — the economic cut Section 9.9 derives — the flag list grows to six as C002 and C010 join: TP 3, FP 3, FN 0, TN 4; accuracy 0.70; precision 0.50; recall 1.00; F1 ≈ 0.67.

After you run: reconcile every printed figure against your hand arithmetic, then notice the result the next cell will price. The majority-class baseline — treat nobody — is 7 ÷ 10 = 0.70 accurate. The model at either threshold is *also* 0.70 accurate. Three policies, one accuracy, and nothing in that number distinguishes them. Write the sentence in your notebook before continuing: accuracy has just failed to detect a difference the program will pay for.

Investigate if: your counts disagree with the printed ones anywhere. Every number in this cell is reproducible with a pencil, and a mismatch is arithmetic rather than convention.

The miniature's payoff is not the matrices; it is what happens when the four counts meet the ledger. Code 9.2 prices five policies on the same ten customers, using the Section 9.9 figures — twelve dollars per treatment, forty dollars of expected benefit per treated true churner — and it handles the empty denominator explicitly, because a do-nothing policy treats nobody and precision is undefined rather than zero.

**Code 9.2. Price five policies on one ledger**

```
COST, BENEFIT = 12.0, 40.0   # per treatment; per treated true churner

def price(name, treat, truth=mini["churned"]):
    """Confusion counts and expected net value for any treatment vector.
    Precision is undefined -- not zero -- when nothing is treated."""
    treat = pd.Series(treat, index=truth.index).astype(int)
    tp = int(((treat == 1) & (truth == 1)).sum())
    fp = int(((treat == 1) & (truth == 0)).sum())
    fn = int(((treat == 0) & (truth == 1)).sum())
    tn = int(((treat == 0) & (truth == 0)).sum())
    treated = tp + fp
    return {"policy": name, "TP": tp, "FP": fp, "FN": fn, "TN": tn,
            "treated": treated,
            "accuracy": (tp + tn) / len(truth),
            "precision": tp / treated if treated else float("nan"),
            "recall": tp / (tp + fn) if (tp + fn) else float("nan"),
            "net": tp * BENEFIT - treated * COST}

policies = pd.DataFrame([
    price("treat nobody",        0),
    price("model at 0.50",  (mini["churn_prob"] >= 0.50).astype(int)),
    price("model at 0.30",  (mini["churn_prob"] >= 0.30).astype(int)),
    price("ninety-day rule",     mini["lapsed_90d"]),
    price("treat everyone",      1),
]).set_index("policy")

print(policies.round(2).to_string())
```

*Expected output: one five-row table carrying the four confusion counts, the treated count, accuracy, precision, recall, and expected net value for every policy — with precision printed as NaN for the do-nothing row rather than raising a division error.*

Input: the miniature's scores, truths, and incumbent flag. Transformation: five treatment vectors run through one pricing function. Output: the exhibit the whole chapter turns on, at a scale where every cell is checkable by hand. Writing the pricing as one function rather than five blocks is the point of the cell's construction: every policy is graded by identical arithmetic, which is the miniature version of Section 8.9's same-metric clause.

**VERIFICATION CHECK**

Before you run: price all five policies with a pencil. Treat nobody: nothing spent, nothing saved, net $0. The 0.50 cut treats four customers for $48 and reaches two churners worth $80, netting +$32. The 0.30 cut treats six for $72 and reaches all three for $120, netting +$48. The ninety-day rule treats C004, C006, C009, and C010 — four customers for $48 — and catches one churner worth $40, netting −$8. Treat everyone treats ten for $120 and reaches all three for $120, netting exactly $0 — the CFO's instinct, confirmed to the dollar.

After you run: reconcile all five nets, then write the two sentences the miniature has earned. First, accuracy cannot distinguish policies whose economics differ by forty-eight dollars on ten customers: three rows post exactly 0.70 while netting $0, +$32, and +$48, and across all five policies the spread runs from −$8 to +$48 with no help from the accuracy column. Second, the threshold — not the model — is what moved the program from +$32 to +$48, because the same scores were cut in a different place. Then identify the single customer whose reclassification between the two thresholds contributes most to the improvement, and say in one sentence what her profile implies about which customers live near a threshold.

Investigate if: the ninety-day rule's net is not negative. On these ten customers the incumbent treats three loyal customers and one churner, which is exactly the failure Section 9.1 predicted from Chapter 5's lumpy cadence — and a positive net means the flag column was transcribed incorrectly.

### *9.12.2 Lab 9.1, Part B: The Frame, the Label, and the Sealed Split*

Part B builds the real thing on Chapter 8's frame. Begin, per the standing rule, with the frame in a markdown cell: the churn definition of Section 9.3 in its four-element form, both eligibility conditions, the declared selection protocol (stratified five-fold cross-validation inside the training customers), the declared baselines (constant score, continuous recency, and the ninety-day rule as a fixed policy), and the split protocol with its seed. Only then does code run. Code 9.3 rebuilds the snapshot feature table using the same parameterized builder Chapter 8 wrote, so that nothing in this chapter's pipeline depends on a notebook the reader may not have open.

**Code 9.3. Rebuild the snapshot frame from Chapter 8**

```
import numpy as np
import pandas as pd

# The companion repository ships Chapter 8's Codes 8.2 and 8.3 as a
# module. Students who prefer to read the source can paste those two
# cells instead; the columns are identical either way.
from stylecraft_features import build_customer_features

FEATURE_START = pd.Timestamp("2024-07-01")
SNAPSHOT      = pd.Timestamp("2025-12-31")
OUTCOME_START = pd.Timestamp("2026-01-01")
OUTCOME_END   = pd.Timestamp("2026-06-30")

# Certified files are loaded per Appendix A as transactions_clean,
# customers, products, and stores.
cf_snap = build_customer_features(transactions_clean, customers, products,
                                  snapshot=SNAPSHOT,
                                  window_start=FEATURE_START)

tx = transactions_clean.assign(
    order_date=transactions_clean["order_date"].dt.normalize())
tx_feat = tx[tx["order_date"].between(FEATURE_START, SNAPSHOT,
                                      inclusive="both")]
tx_out  = tx[tx["order_date"].between(OUTCOME_START, OUTCOME_END,
                                      inclusive="both")]

# Both ends of both windows, asserted rather than assumed (Section 8.3).
assert tx_feat["order_date"].min() >= FEATURE_START
assert tx_feat["order_date"].max() <= SNAPSHOT
assert tx_out["order_date"].min()  >= OUTCOME_START
assert tx_out["order_date"].max()  <= OUTCOME_END

# ELIGIBILITY, both conditions evaluable at the snapshot (Section 9.3).
signed_up = set(customers.loc[customers["signup_date"] <= SNAPSHOT,
                              "customer_id"])
purchased = set(cf_snap.index)
eligible  = sorted(signed_up & purchased)
model_df  = cf_snap.loc[eligible].copy()

print("customers in the file:                    ", len(customers))
print("signed up by the snapshot:                ", len(signed_up))
print("purchased inside the feature window:      ", len(purchased))
print("eligible (both conditions) -> model_df:   ", len(model_df))
print("signed up, never purchased (activation):  ",
      len(signed_up - purchased))
print("acquired after the snapshot (out of scope):",
      len(customers) - len(signed_up))
```

*Expected output: six counts that reconcile against the customer file, and four window assertions passing silently.*

Input: the certified transaction, customer, product, and store files. Transformation: one parameterized feature build at the snapshot, two window filters with both ends enforced, and a two-condition eligibility intersection. Output: model_df, one row per eligible customer, carrying only snapshot-safe features — and a printed partition of the customer file that names the

excluded groups rather than dropping them silently. The activation population in the fifth line is the group Section 9.3 removed on purpose: customers who signed up and never bought, for whom recency, average order value, discount share, and store share are all undefined. They are a real marketing problem and a different one.

**Code 9.4. Construct the churn label from purchase activity**

```
# The label follows the DEFINITION -- no completed purchase order in the
# outcome window -- not the convenience of a zero revenue total.
outcome_orders = tx_out.groupby("customer_id")["order_id"].nunique()
model_df["outcome_orders"] = outcome_orders.reindex(model_df.index,
                                                    fill_value=0)
model_df["churned"] = model_df["outcome_orders"].eq(0).astype(int)

# Chapter 8's spend label, kept for reconciliation ONLY. It is not the
# label and it is never a feature.
outcome_spend = tx_out.groupby("customer_id")["line_revenue"].sum()
model_df["spend_next6m"] = outcome_spend.reindex(model_df.index,
                                                 fill_value=0.0)

print("eligible customers:", len(model_df))
print("base rate (churned):", round(model_df["churned"].mean(), 3))

# Reconciliation 1: activity vs. net revenue. These need NOT agree.
zero_spend = model_df["spend_next6m"].le(0)
disagree = int((zero_spend != model_df["churned"].astype(bool)).sum())
print("customers where zero net revenue and zero purchase activity "
      f"disagree: {disagree}")
if disagree:
    print(model_df.loc[zero_spend != model_df["churned"].astype(bool),
                       ["outcome_orders", "spend_next6m"]].head())

# Reconciliation 2: the shipped answer key (dataset spec, Section 4).
cf_key = pd.read_csv("customer_features.csv").set_index("customer_id")
agreement = model_df["churned"].eq(
    cf_key["churn_label"].reindex(model_df.index)).mean()
print("agreement with the shipped label:", round(agreement, 4))
```

*Expected output: the eligible count, the base rate, the count of customers whose activity and revenue definitions disagree (with a sample of them if any exist), and the agreement rate against the shipped answer key.*

Input: the outcome-window transactions and the eligible feature table. Transformation: a distinct-order count per customer, reindexed onto the eligible population so that a customer with no outcome-window rows receives zero rather than a missing value. Output: the churn flag, plus two reconciliations the deliverable reports. The reindex-with-fill is the cell's load-bearing line, and it is the classification twin of Chapter 8's fillna argument: a customer absent from the outcome-window groupby is not missing data, she is the positive class.

**VERIFICATION CHECK**

Before you run: write three predictions. The base rate — from Chapter 5's certified repeat-purchase findings, predict the share of eligible customers with no outcome-window purchase; the designed expectation is in the neighborhood of a third, and a printed value far from your prediction is a label bug until explained. The disagreement count between the activity definition and a net-revenue definition — predict whether it is zero, and write down what a non-zero count would tell you about the revenue column. And the dependency: confirm in writing that the label was built from outcome-window data alone and every feature from feature-window data alone, the wall of Section 8.3 restated once per chapter because every leak in this chapter's history walked through it.

After you run: grade all three. If the disagreement count is non-zero, open the sampled rows and classify each: a zero-value order, a fully returned order, or a negative net. Then write the sentence the deliverable needs on its first page — whether, on this certified file, the churn flag happens to coincide with the complement of Chapter 8's spend label, and why the chapter still refuses to define it that way. If agreement with the shipped key is not exactly 1.0, find a disagreeing customer and determine which definitional lever of Table 9.1 the two labels moved differently; that forensic exercise is worth more than the agreement figure.

Investigate if: the base rate lands above one half. Under this six-month definition it should not, and a majority-churn file usually means the outcome window was filtered incorrectly or the eligibility intersection was skipped, admitting never-activated signups who are churned by construction.

**Code 9.5. Split the eligible population and seal the test set**

```
from sklearn.model_selection import train_test_split

# Chapter 8's snapshot-safe columns, minus monetary: by Table 6.5's own
# derivation rules monetary equals aov times frequency, so including all
# three walks straight into Section 7.10's multicollinearity trap.
FEATURES = ["recency_days", "frequency", "tenure_days", "aov",
            "orders_per_month", "revenue_per_month", "discount_share",
            "store_share", "occasionwear_share", "app_user",
            "email_opt_in"]

assert set(FEATURES) <= set(model_df.columns), "a feature does not exist"
for forbidden in ["churned", "outcome_orders", "spend_next6m"]:
    assert forbidden not in FEATURES, f"{forbidden} was handed over"

train_df, test_df = train_test_split(
    model_df, test_size=0.20, random_state=42,
    stratify=model_df["churned"])

train_ids, test_ids = train_df.index, test_df.index
assert train_ids.intersection(test_ids).empty
assert len(train_ids) + len(test_ids) == len(model_df)

X_tr, y_tr = train_df[FEATURES], train_df["churned"]

# Development diagnostics come from the TRAINING customers only. The test
# base rate is deliberately NOT printed: knowing it would leak forward
# into every expectation about what a plausible result looks like.
print("training customers:", len(train_ids))
print("test customers:    ", len(test_ids))
print("training base rate:", round(y_tr.mean(), 3))
print("training majority-class accuracy:",
      round(max(y_tr.mean(), 1 - y_tr.mean()), 3))
print("\nTEST SET SEALED. Its outcomes stay unopened until Code 9.14.")
```

*Expected output: two population counts, the training base rate and its majority-class floor, and the sealing notice; four structural assertions pass silently.*

Input: the labeled eligible table. Transformation: a seeded, stratified 80/20 split at the customer grain. Output: two frames and two index objects that every later cell reuses. Two details enforce disciplines this chapter argued for. The stratify argument is Section 9.6's one-line concept-level answer to imbalance: both sides of the split inherit the true base rate, every fold inherits it too, and nothing about the probability scale is distorted the way rebalancing would distort it. And the cell prints the *training* base rate and floor rather than the test ones, because a sealed test set is not sealed once its outcomes have been seen; the test figures appear in Code 9.14, after the grade is recorded, where the same numbers are context for reading a result rather than an influence on producing one.

### *9.12.3 Lab 9.1, Part C: Reading the Risk in Odds*

Part C reads a model before any model is raced, because Section 9.4's odds-ratio skill needs a model whose coefficients are in customer units and it will not get one from the leaderboard. The

candidates of Lab 9.2 standardize their inputs and carry scikit-learn's default regularization, which is right for prediction and wrong for reading: a coefficient on standardized recency is a statement about standard deviations, and a penalized coefficient is shrunk toward zero by an amount nobody in the meeting agreed to. Code 9.6 therefore fits a separate, deliberately small model whose only job is interpretation — the explanation-versus-prediction boundary of Section 8.2, drawn inside one chapter's lab rather than across two.

**Code 9.6. Fit and read an interpretive logistic model**

```
import statsmodels.formula.api as smf

# AN INTERPRETATION EXERCISE, NOT A CANDIDATE. This model is fitted on
# the training customers in original units and is never scored, never
# entered on the leaderboard, and never used to treat anyone. Its job is
# to make Section 9.4's odds-ratio sentences checkable.
interpretive = smf.logit(
    "churned ~ recency_days + frequency + tenure_days"
    " + app_user + email_opt_in + discount_share + store_share",
    data=train_df).fit(disp=False)

print(interpretive.summary().tables[1])

odds = np.exp(interpretive.params).rename("odds_ratio")
ci = np.exp(interpretive.conf_int()).rename(columns={0: "lo", 1: "hi"})
print("\n" + pd.concat([odds, ci], axis=1).round(3).to_string())

b = interpretive.params["recency_days"]
print(f"\nper additional day of silence, churn odds x {np.exp(b):.4f}")
print(f"over 90 additional days,        churn odds x {np.exp(90*b):.2f}")
```

*Expected output: the coefficient table with standard errors, z statistics, and p-values; a second table giving each exponentiated coefficient with its confidence interval as an odds ratio; and the recency effect translated to one day and to ninety days.*

Input: the training customers only. Transformation: one maximum-likelihood fit on seven predictors in their native units. Output: odds ratios a meeting can hear. Three design choices carry the section's argument. The feature list is shorter than the predictive one, because a model built to be read should contain only variables whose coefficients someone is prepared to defend. The fit is on training customers rather than on everything, because a chapter that seals its test set does not open it for a side exercise. And the model is labeled in the comment as ineligible for the race, because the fastest way to lose the explanation-versus-prediction boundary is to let one artifact quietly play both roles.

**VERIFICATION CHECK**

Before you run: write the predicted sign of every coefficient from the designed churn structure and Chapter 5's findings — recency positive (silence raises churn odds), frequency negative, tenure negative, the two engagement flags negative, discount share positive, store share positive. Then predict, to one decimal place, the odds multiplier for ninety additional days of silence.

After you run: grade every sign, then translate two coefficients into plain odds language per Section 9.4 — one continuous ("each additional day of silence multiplies churn odds by …") and one flag ("app users carry … times the churn odds of comparable non-users") — and check each sentence's verb against Section 7.2's registry. Add the ceteris paribus clause explicitly, and then write the sentence that limits it: because of the S-curve, the same odds ratio moves a mid-risk customer's probability substantially and a low-risk customer's barely at all, so any statement about percentage points needs a stated starting probability.

Investigate if: a sign contradicts the designed structure, or a confidence interval for an odds ratio comfortably straddles 1.0 for a variable the design says matters. The first is a data or formula problem worth tracing; the second is honest evidence of a weak effect, and the deliverable should report it as such rather than quoting the point estimate alone.

### *9.12.4 Lab 9.2, Part A: Choosing Flexibility Inside the Training Data*

Chapter 8's Part B made the U-shape visible without consulting the test set, and this lab owes the same discipline to two settings this chapter introduced: the tree's maximum depth and k-NN's k. Both are chosen here, from training-only evidence, with cross-validated AUC as the criterion and training AUC printed beside it so the overfitting signature of Table 8.4 is visible rather than inferred. The selection is not a bare maximum. Section 8.8 argued that flexibility must earn its place, and Code 9.7 makes that argument executable with a one-standard-error rule: find the best mean cross-validated AUC, define an acceptable band one standard error below it, and inside that band take the *simplest* candidate — the shallowest tree, and the largest k, since a larger neighborhood is a smoother and less flexible model.

**Code 9.7. Select the tree's depth inside the training folds**

```
from sklearn.model_selection import StratifiedKFold, cross_validate
from sklearn.tree import DecisionTreeClassifier
from sklearn.neighbors import KNeighborsClassifier
from sklearn.preprocessing import StandardScaler
from sklearn.pipeline import make_pipeline
from sklearn.metrics import roc_auc_score

# ONE fold generator, built once, shared by every candidate and every
# selection decision in this lab: the same-split clause of Section 8.9.
CV = StratifiedKFold(n_splits=5, shuffle=True, random_state=42)

def auc_scorer(estimator, X, y):
    return roc_auc_score(y, estimator.predict_proba(X)[:, 1])

def one_se_band(table):
    """Rows whose mean CV AUC is within one standard error of the best.
    This is what turns 'prefer the simpler model when two are
    indistinguishable' from advice into an algorithm."""
    best = table["cv_auc"].idxmax()
    se = table.loc[best, "cv_sd"] / np.sqrt(CV.get_n_splits())
    return table[table["cv_auc"] >= table.loc[best, "cv_auc"] - se]

rows = []
for depth in [2, 3, 4, 5, 6, 8, 12, 0]:      # 0 means "no depth limit"
    md = None if depth == 0 else depth
    tree = DecisionTreeClassifier(max_depth=md, random_state=42)
    s = cross_validate(tree, X_tr, y_tr, cv=CV,
                       scoring={"auc": auc_scorer},
                       return_train_score=True)
    rows.append({"max_depth": "none" if md is None else str(md),
                 "complexity": 99 if md is None else md,
                 "train_auc": s["train_auc"].mean(),
                 "cv_auc": s["test_auc"].mean(),
                 "cv_sd": s["test_auc"].std(),
                 "gap": s["train_auc"].mean() - s["test_auc"].mean()})
tree_flex = pd.DataFrame(rows).set_index("max_depth")
print(tree_flex.round(3).to_string())

# Simplest candidate inside the one-standard-error band, not the maximum.
tree_band = one_se_band(tree_flex)
depth_label = tree_band["complexity"].idxmin()
BEST_DEPTH = None if depth_label == "none" else int(depth_label)
print(f"\n{len(tree_band)} depths within one standard error of the best"
      f" -> simplest is depth {depth_label}")
```

*Expected output: an eight-row table carrying a complexity rank, mean training AUC, mean cross-validated AUC, the fold-to-fold standard deviation, and the training-minus-cross-validated gap; then how many depths fell inside the one-standard-error band and which shallowest member was selected.*

Input: the training customers only. Transformation: eight tree fits, each cross-validated on the shared stratified folds, then one selection rule applied to the table. Output: the flexibility curve of Section 8.8 in classification units, and a depth chosen by a rule rather than by an eye. Read the gap column first: it is the overfitting signature, and the unlimited tree should show the widest gap

in the table — training AUC near 1.0, cross-validated AUC well below the best constrained depth. That is the demonstration the draft of this lab performed against the test set, and it is strictly better performed here, because the test set is not a diagnostic instrument and a demonstration that spends it has bought a lesson with the deliverable's only honest grade.

The same rule now chooses k, on the same folds and with the same band. One detail flips: the complexity column carries a negative sign for k-NN, because the two dials run in opposite directions. A deeper tree is more flexible; a *larger* neighborhood is less flexible, since it averages over more customers and drifts toward the base rate. The simplest candidate inside the band is therefore the shallowest tree and the largest k.

**Code 9.8. Select k inside the same folds**

```
rows = []
for k in [5, 15, 25, 50, 100, 200]:
    knn = make_pipeline(StandardScaler(),
                        KNeighborsClassifier(n_neighbors=k))
    s = cross_validate(knn, X_tr, y_tr, cv=CV,
                       scoring={"auc": auc_scorer},
                       return_train_score=True)
    rows.append({"k": k, "complexity": -k,   # larger k = simpler model
                 "train_auc": s["train_auc"].mean(),
                 "cv_auc": s["test_auc"].mean(),
                 "cv_sd": s["test_auc"].std(),
                 "gap": s["train_auc"].mean() - s["test_auc"].mean()})
knn_flex = pd.DataFrame(rows).set_index("k")
print("\n" + knn_flex.round(3).to_string())

knn_band = one_se_band(knn_flex)
BEST_K = int(knn_band["complexity"].idxmin())
print(f"\n{len(knn_band)} values of k within one standard error of the"
      f" best -> simplest (largest k) is k = {BEST_K}")
```

*Expected output: a six-row table with the same five columns, then the size of the one-standard-error band and the selected value of k.*

Input and transformation as above, with six neighborhood sizes in place of eight depths. Output: the second of the two settings the leaderboard needs, chosen by the same declared rule. Both cells share one fold generator and one band function, which is what makes the two selections comparable and what keeps the same-split clause of Section 8.9 true across them.

**VERIFICATION CHECK**

Before you run: predict three shapes. Training AUC as depth increases — state the direction and whether it is guaranteed or merely expected. Cross-validated AUC as depth increases — predict where the turn falls, and write down what you will conclude if no turn appears within the tested range. And k-NN at k = 5 against k = 200 — say which end you expect to overfit, which end you expect to collapse toward the base rate, and why the second is the majority-class baseline rediscovered as a limiting case. Then predict how many candidates will fall inside the one-standard-error band.

After you run: grade every prediction, then check the rule's work by hand. Confirm that the selected depth is the shallowest inside the band rather than the highest scoring, and say in one sentence what the rule bought: a model whose behavior is easier to explain, monitor, and retrain, at a cost in cross-validated AUC that the band declares immaterial. Name the two sides of each curve in Section 8.8's vocabulary and state what the unlimited tree and k = 5 are actually memorizing.

Investigate if: the band contains every candidate. That means the fold-to-fold noise swamps every difference between settings, which is a legitimate finding on a small training population — report it, take the simplest candidate, and say plainly that the flexibility dial did not matter here.

### *9.12.5 Lab 9.2, Part B: The Training-Only Leaderboard*

The leaderboard was fixed in the frame cell: cross-validated AUC on shared stratified folds, five candidates, one fold generator, a selection rule written before the race. One change from the draft of this lab is worth stating in advance, because it decides what the leaderboard can and cannot conclude. The leaderboard grades **ranking only**. Expected net value does not appear on it, because a dollar figure computed from raw model scores is denominated in probabilities nobody has checked, and Section 9.8's own rule forbids that. The economics arrive in Part C, once the finalist has been calibrated inside the training data, and they arrive as a gate the selected model must pass rather than as a column models compete on.

The two baselines are candidates rather than commentary — fitted, scored, and ranked by the identical call, because a baseline evaluated by a different route is not evidence about the same question — and both now come from the library rather than from hand-written estimator classes. The constant-score floor is scikit-learn's `DummyClassifier` predicting the training prior for everyone; its AUC is 0.5 by construction, and printing it is a check that the scoring plumbing works. The continuous-recency baseline is a logistic regression on recency alone. That choice is deliberate and repairs a real defect in the obvious alternative: a baseline that scores customers by rescaling recency between the smallest and largest values *in the data being scored* would define its own scale from the evaluation population, so a customer's score would depend on who happened to be scored alongside her, and every fold would use a different ruler. A recency-only logistic model fits its parameters on the training data like any other model, produces genuine probabilities that can be checked for calibration, and — if it turns out to be the best thing available — can honestly carry the economic threshold.

The ninety-day rule itself is not on this leaderboard, and the omission is deliberate. A binary 0/1 flag has only one nontrivial operating point, so its AUC is not comparable to the AUC of a continuous score in the way a shared column implies; the rule is evaluated as a fixed policy in Part C and Part E, on the confusion matrix, precision, recall, treated count, and expected net value that are the terms it actually competes in.

**Code 9.9. Register the candidates and declare the selection rule**

```
from sklearn.dummy import DummyClassifier
from sklearn.linear_model import LogisticRegression

COST, BENEFIT = 12.0, 40.0
THRESHOLD = COST / BENEFIT          # 0.30, derived in Section 9.9
RECENCY_ONLY = ["recency_days"]

constant_floor = DummyClassifier(strategy="prior")
recency_logit  = make_pipeline(StandardScaler(),
                               LogisticRegression(max_iter=1000))
logistic = make_pipeline(StandardScaler(),
                         LogisticRegression(max_iter=1000))
tree = DecisionTreeClassifier(max_depth=BEST_DEPTH, random_state=42)
knn  = make_pipeline(StandardScaler(),
                     KNeighborsClassifier(n_neighbors=BEST_K))

# Each candidate is paired with the columns it is allowed to see -- which,
# read the other way, is a written record of what it is forbidden to see.
BASELINE = "recency-only logistic (baseline)"
CANDIDATES = {
    "constant score (floor)":  (constant_floor, FEATURES),
    BASELINE:                  (recency_logit,  RECENCY_ONLY),
    "logistic regression":     (logistic,       FEATURES),
    f"tree (depth {BEST_DEPTH})": (tree,        FEATURES),
    f"k-NN (k = {BEST_K})":    (knn,            FEATURES),
}

# THE SELECTION RULE, declared here -- before the race is scored -- and
# executed unedited in Code 9.11 and Code 9.13.
MIN_AUC_MARGIN = 0.02      # AUC a model must add over the baseline
MIN_FOLDS_WON  = 4         # of five, on paired per-fold differences
SIMPLICITY_ORDER = ["logistic regression", f"tree (depth {BEST_DEPTH})",
                    f"k-NN (k = {BEST_K})"]
MAX_CALIBRATION_ERROR = 0.05   # near the threshold, on out-of-fold scores

for name, (_, cols) in CANDIDATES.items():
    assert set(cols) <= set(train_df.columns), name
    for forbidden in ("churned", "outcome_orders", "spend_next6m"):
        assert forbidden not in cols, f"{name} was handed {forbidden}"
print("candidates registered:", len(CANDIDATES),
      "| economic threshold:", THRESHOLD)
```

*Expected output: the candidate count and the derived threshold of 0.3, after fifteen structural assertions confirm that every candidate's column list exists and that none of them contains the label or its ingredients.*

Input: the two settings chosen in Code 9.7. Transformation: two library baselines and three pipelines, paired with their permitted columns. Output: a registry, and a selection rule written before any candidate is scored. That last object is the cell's real content: Section 8.9's leaderboard discipline is enforceable only if the rule predates the results, and a rule typed after the table has been read is a preference wearing a rule's clothes. Note that the rule now has four clauses rather than three — an AUC margin, a folds-won requirement, a simplicity order, and a calibration tolerance — because Section 8.8 made calibration a condition on the economics rather than a nicety.

**Code 9.10. Build the training-only ranking leaderboard**

```
rows, fold_auc = [], {}
for name, (est, cols) in CANDIDATES.items():
    s = cross_validate(est, train_df[cols], y_tr, cv=CV,
                       scoring={"auc": auc_scorer})
    fold_auc[name] = s["test_auc"]
    rows.append({"candidate": name,
                 "features": len(cols),
                 "cv_auc": s["test_auc"].mean(),
                 "auc_sd": s["test_auc"].std()})

leaderboard = pd.DataFrame(rows).set_index("candidate")
leaderboard["auc_margin_vs_baseline"] = (
    leaderboard["cv_auc"] - leaderboard.loc[BASELINE, "cv_auc"])
print(f"training base rate: {y_tr.mean():.3f}   "
      f"majority-class accuracy: {max(y_tr.mean(), 1 - y_tr.mean()):.3f}")
print(leaderboard.round(3).to_string())
assert abs(leaderboard.loc["constant score (floor)",
                           "cv_auc"] - 0.5) < 0.01
print("\nNo dollar figure appears on this table by design: expected value"
      "\nis computed in Code 9.12, from calibrated probabilities.")
```

*Expected output: the base rate and majority-class floor in a header line, then a five-row table of feature count, cross-validated AUC, its fold-to-fold spread, and each candidate's AUC margin over the recency-only baseline; one assertion confirms the constant scorer lands at 0.5, and a closing note records why no net-value column is present.*

Input: the training customers only. Transformation: five-fold stratified cross-validation of every candidate on one shared fold generator, each scored on the columns its registry entry permits. Output: the ranking evidence, and the only evidence the analyst is permitted to use when choosing a model family. Read the standard-deviation column as seriously as the mean, exactly as Chapter 8 instructed: a margin smaller than the fold-to-fold spread of the metric it is measured on is not yet a finding.

**VERIFICATION CHECK**

Before you run: write four predictions. Where will the recency-only baseline's AUC land, and why is a value well above 0.5 the *expected* result rather than a disappointment? Which of the three multivariable models wins, and by how much over the baseline? Is that margin larger than the fold-to-fold spread? And what would an implausibly high AUC look like on this base, given the volatility the label's construction implies?

After you run: grade every prediction, then audit your own table against Section 8.9's three clauses. Same split: confirm that one fold generator produced every row. Same metric: one column, declared in advance, with no dollar figure available to switch to. Same baseline: state each model's margin over the recency-only baseline and say whether it exceeds the spread. Then write the sentence this leaderboard licenses and the sentence it does not: it licenses a claim about which family orders customers best, and it licenses nothing at all about money.

Investigate if: any candidate's cross-validated AUC lands near or above 0.9. Against a base whose churn label summarizes six months of individual human behavior, that is Section 8.4's too-good signature, and the feature trace comes before the celebration — starting with the recency range check of Section 8.12, because a recency below 181 days at a December 31 snapshot is arithmetically impossible.

### *9.12.6 Lab 9.2, Part C: The Finalist, Calibrated, and the Economic Gate*

Part C is where the chapter's own rule about calibration becomes a step in the pipeline rather than a paragraph of advice. The sequence has six moves and the order is the argument: choose the family from training-only ranking evidence; name the provisional finalist; calibrate *that* finalist entirely inside the training population; obtain out-of-fold calibrated probabilities; evaluate reliability and expected net value on those probabilities; and only then decide whether the program is fundable. Nothing in that sequence touches the test set, and every dollar figure it produces comes from probabilities that were checked before they were multiplied.

**Code 9.11. Apply the selection rule and calibrate the finalist**

```
from sklearn.base import clone
from sklearn.calibration import CalibratedClassifierCV

base = fold_auc[BASELINE]
rows = []
for name in SIMPLICITY_ORDER:
    diff = fold_auc[name] - base
    rows.append({"candidate": name,
                 "mean paired AUC margin": diff.mean(),
                 "sd of paired margins": diff.std(),
                 "folds won": int((diff > 0).sum()),
                 "clears the rule": bool(diff.mean() >= MIN_AUC_MARGIN
                                         and (diff > 0).sum()
                                         >= MIN_FOLDS_WON)})
paired = pd.DataFrame(rows).set_index("candidate")
print(paired.round(4).to_string())

qualified = [n for n in SIMPLICITY_ORDER
             if paired.loc[n, "clears the rule"]]
FALLBACK = not qualified
FINALIST = qualified[0] if qualified else BASELINE
print("\nqualifying, simplest first:", qualified or "none")
print("FINALIST:", FINALIST)
if FALLBACK:
    print("No multivariable model cleared the declared rule. The finalist"
          "\nis the recency-only baseline -- itself a fitted probability"
          "\nmodel, so it can be calibrated and thresholded honestly, but"
          "\nthe deliverable's recommendation changes: see Code 9.12.")

# Calibration is fitted INSIDE the training population, on predictions the
# underlying classifier did not make about its own fitting data
# (scikit-learn developers, 2026a). This object -- estimator PLUS
# calibrator -- is what gets frozen and what gets graded.
finalist_estimator, FINALIST_COLS = CANDIDATES[FINALIST]
final_model = CalibratedClassifierCV(clone(finalist_estimator),
                                     cv=CV, method="isotonic")
print("\nfinalist columns:", len(FINALIST_COLS),
      "| calibrator: isotonic, fitted inside the training folds")
```

*Expected output: a three-row paired-margin table with the folds each candidate won and whether it clears the declared rule; the qualifying list in simplicity order; the named finalist; the fallback notice if no multivariable model qualified; and a confirmation of the finalist's feature count and calibration method.*

Input: the per-fold AUCs stored by Code 9.10. Transformation: fold-by-fold subtraction against the baseline, one rule executed unedited, and one calibration wrapper. Output: a finalist arrived at by a rule rather than a preference, and — crucially — a finalist that is now a *pair*, estimator plus calibrator, which is the object every later cell scores, freezes, and deploys. The comparison is paired for the reason Section 8.9 gave: folds differ in difficulty, a fold holding an unusual mix of lumpy occasion shoppers is hard for every candidate at once, and subtracting the baseline fold by fold cancels the shared difficulty that each candidate's own standard deviation contains.

**Code 9.12. Build the out-of-fold reliability exhibit**

```
import matplotlib.pyplot as plt
from sklearn.model_selection import cross_val_predict

def net_per_1000(y_true, p, threshold=THRESHOLD):
    """Expected net value per 1,000 evaluated customers, so that folds
    and populations of different sizes stay comparable."""
    y_true = np.asarray(y_true)
    treat = (np.asarray(p) >= threshold).astype(int)
    tp = int(((treat == 1) & (y_true == 1)).sum())
    treated = int(treat.sum())
    return 1000.0 * (tp * BENEFIT - treated * COST) / len(y_true)

# Out-of-fold probabilities from the CALIBRATED pair. Because final_model
# calibrates internally, this is a nested loop: the calibrator of each
# outer fold never sees that fold's held-out customers.
X_fin = train_df[FINALIST_COLS]
oof = cross_val_predict(final_model, X_fin, y_tr, cv=CV,
                        method="predict_proba")[:, 1]

edges = np.linspace(0.0, 1.0, 11)
cal = (pd.DataFrame({"p": oof, "y": y_tr.to_numpy()})
       .assign(bin=lambda d: pd.cut(d["p"], edges, include_lowest=True))
       .groupby("bin", observed=True)
       .agg(customers=("y", "size"),
            mean_predicted=("p", "mean"),
            observed_rate=("y", "mean")))
cal["difference"] = cal["observed_rate"] - cal["mean_predicted"]
print(cal.round(3).to_string())

fig, ax = plt.subplots(figsize=(4.6, 4.6))
ax.plot([0, 1], [0, 1], linestyle="--", linewidth=1,
        label="perfect calibration")
ax.plot(cal["mean_predicted"], cal["observed_rate"], marker="o",
        label="out of fold, training customers")
ax.axvline(THRESHOLD, linewidth=1, linestyle=":",
           label=f"economic threshold {THRESHOLD:.2f}")
ax.set_xlabel("mean predicted churn probability")
ax.set_ylabel("observed churn rate")
ax.set_title("Reliability of the calibrated finalist")
ax.legend(fontsize=8)
plt.tight_layout(); plt.show()
```

*Expected output: a ten-row reliability table carrying each bin's customer count, mean predicted probability, observed churn rate, and their difference; then a reliability plot with the diagonal and the economic threshold marked.*

Input: the calibrated finalist and the training customers. Transformation: nested cross-validated probability predictions, binned against observed rates. Output: the evidence that entitles Section 9.9's arithmetic to be applied. Three details are deliberate. The probabilities are out of fold and come from the calibrated pair, so the plot describes the object that will actually be deployed rather than an uncalibrated ancestor of it. Because `final_model` calibrates internally, the call is a nested loop: the calibrator inside each outer fold never sees that fold's held-out customers. And the customer count per bin is printed because a bin holding eleven customers

cannot support a calibration verdict, and a reliability plot without counts invites confident conclusions from empty bins.

The exhibit is evidence; the next cell is the decision. Its two conditions were declared in Code 9.9, both are computed from training customers only, and both must hold before any money is discussed.

**Code 9.13. Clear the economic gate**

```
# THE GATE. Two conditions, both declared in Code 9.9, both computed from
# training customers only, both required before any money is discussed.
left  = np.array([b.left for b in cal.index], dtype=float)
right = np.array([b.right for b in cal.index], dtype=float)
near = cal[(left >= THRESHOLD - 0.2) & (right <= THRESHOLD + 0.2)]
worst = (float(near["difference"].abs().max())
         if len(near) else float("nan"))

model_net = net_per_1000(y_tr, oof)
incumbent = (train_df["recency_days"] > 90).astype(float)
incumbent_net = net_per_1000(y_tr, incumbent, threshold=0.5)

print(f"\nworst calibration deviation near the threshold: {worst:.3f}"
      f"  (tolerance {MAX_CALIBRATION_ERROR})")
print(f"cross-validated net per 1,000, calibrated:  ${model_net:,.0f}")
print(f"cross-validated net per 1,000, incumbent:   "
      f"${incumbent_net:,.0f}")

GATE_PASSED = (worst <= MAX_CALIBRATION_ERROR
               and model_net > max(0.0, incumbent_net))
print("\nECONOMIC GATE:", "PASSED" if GATE_PASSED else "NOT PASSED")
if not GATE_PASSED:
    print("The scored program is not yet fundable on training evidence."
          "\nHonest responses, in order: recalibrate or refit inside the"
          "\ntraining folds; retain the ninety-day incumbent policy;"
          "\nrecommend rank-based use with no probability threshold; or"
          "\nreturn to framing and features. Do NOT open the test set"
          "\nlooking for a better number.")
```

*Expected output: the worst calibration deviation near the threshold against its declared tolerance, the calibrated model's and the incumbent's cross-validated net value per thousand customers, and the gate's verdict with its consequences.*

Input: the out-of-fold calibrated probabilities and the training customers. Transformation: two declared tests — a calibration tolerance near the threshold, and an expected-net-value comparison against zero and against the incumbent. Output: the first place in the lab where a dollar figure legitimately appears, and the decision about whether the scored program is fundable at all. Three details are deliberate. Both tolerances were declared in Code 9.9 rather than chosen by looking at this output, which is what makes this a gate rather than a reaction. The incumbent's net is computed on the same training customers as a fixed policy that fits nothing, so comparing against it here costs nothing and leaks nothing. And the failure branch prints its own honest responses, none of which is "open the test set and look for a better number" — because a gate that can be appealed to the sealed data is not a gate.

**VERIFICATION CHECK**

Before you run: predict the shape of the curve — points above the diagonal mean the model under-predicts churn, below means it over-predicts — and predict whether the calibrated finalist will beat the ninety-day rule on cross-validated net value. Then write the sentence you will owe the deliverable either way: the analytically derived threshold of 0.30 is valid only when the probabilities are sufficiently calibrated for the deployment population.

After you run: read the two bins that straddle 0.30 first, because those are the bins the decision actually depends on; a model beautifully calibrated at 0.05 and off by fifteen points at 0.30 is miscalibrated where it matters. Then state, in one sentence, what would happen to the treated-list size and the expected net if the scores near the cut were systematically ten points too high — and note that AUC would not have moved at all. Finally, record the gate's verdict in the notebook *before* running anything else, because a gate whose result is read after the test set is open is not a gate.

Investigate if: the gate fails. That is a legitimate and reportable outcome, not a failed lab, and the honest responses are printed by the cell itself. Investigate also if the top bins are empty or nearly so: that is ordinary on a base whose churn probabilities concentrate below 0.6, and it means the verdict rests on the lower bins — report the sparsity rather than reading a confident line through three customers.

### *9.12.7 Lab 9.2, Part D: The Sealed Test Set, Opened Once*

Everything so far happened inside the training customers: the family was chosen on ranking evidence, the finalist was calibrated, and the economics were gated. Part D spends the test set — once, on the frozen estimator-plus-calibrator pair.

**Code 9.14. Freeze the model and open the sealed test set once**

```
# Refit the pair on ALL training customers. The cross-validated estimates
# came from models fitted on four folds at a time; this is the object that
# would actually be deployed.
final_model.fit(X_fin, y_tr)

y_te = test_df["churned"]
p_te = final_model.predict_proba(test_df[FINALIST_COLS])[:, 1]

baseline_model = CalibratedClassifierCV(
    clone(CANDIDATES[BASELINE][0]), cv=CV, method="isotonic")
baseline_model.fit(train_df[RECENCY_ONLY], y_tr)
p_base = baseline_model.predict_proba(test_df[RECENCY_ONLY])[:, 1]

tag = "  (fallback -- no model cleared the rule)" if FALLBACK else ""
print(f"finalist: {FINALIST}{tag}")
print(f"training-evidence gate: "
      f"{'PASSED' if GATE_PASSED else 'NOT PASSED'}")
print(f"\ntest base rate:               {y_te.mean():.3f}")
print(f"test majority-class accuracy: "
      f"{max(y_te.mean(), 1 - y_te.mean()):.3f}")
print(f"test AUC, finalist:           {roc_auc_score(y_te, p_te):.3f}")
print(f"test AUC, recency baseline:   {roc_auc_score(y_te, p_base):.3f}")
print("\nTEST SET SPENT. Any further development returns to the folds.")
```

*Expected output: the finalist's name with its fallback and gate status, then four test-set figures — base rate, majority-class accuracy, and two AUCs — followed by the spending notice.*

Input: the training customers, the frozen pair, and the sealed test customers. Transformation: one refit and two scoring passes. Output: the deliverable's central grade. Two details enforce disciplines the chapter argued for. The finalist and the gate verdict are printed *above* the test figures, so the notebook itself records that the decision preceded the grade rather than following it. And the baseline is calibrated too, so the two AUC figures compare like with like and the baseline remains usable as an honest fallback rather than as a scoring convenience.

**VERIFICATION CHECK**

Before you run: write four predictions. How close will the test AUC sit to the finalist's cross-validated AUC, and in which direction do you expect them to differ? What would an alarming result look like in each direction — too poor, and too good? Will the calibrated finalist beat the calibrated recency baseline on the test customers by roughly the margin the folds suggested? And what will you do if it does not?

After you run: place the result on Table 8.4's signature grid, then write the two sentences that survive into the Section 9.13 deliverable: the ranking sentence, which states the finalist's test AUC beside the baseline's, and the floor sentence, which states the test base rate and the majority-class accuracy every later accuracy figure must be read against. Then honor the protocol: the test set is now spent, and any new idea returns to the training folds.

Investigate if: the test AUC sits far above the cross-validated figure. A gap larger than the fold-to-fold spread usually means a column entered the feature list after the leaderboard was frozen, or the feature table was rebuilt at a later snapshot. Both are found by rerunning Code 9.3 from the top and confirming that the row counts and the recency minimum are unchanged.

### *9.12.8 Lab 9.2, Part E: Five Policies, One Table*

Part E is the miniature's strongest lesson at full scale. The threshold was derived before any confusion matrix was printed, in a markdown cell the CFO could read — cost per treatment $12; expected benefit per treated true churner $40 (save rate 0.25 × retained margin $160, both flagged as finance estimates); therefore treat above p = 12 ÷ 40 = 0.30 — and the reliability evidence of Code 9.12 is what entitles that cut to be applied to these scores. One reading rule attaches before the table is printed: if the economic gate of Code 9.12 did not pass, this table is evidence *against* funding the scored program rather than a menu to choose from, and the deliverable reports it that way. Then, and only then, the matrices.

**Code 9.15. Compare five policies on the sealed test customers**

```
scored = test_df.assign(p=p_te)
scored["lapsed_90d"] = (scored["recency_days"] > 90).astype(int)

def price_full(name, treat, truth=scored["churned"]):
    treat = pd.Series(np.asarray(treat), index=truth.index).astype(int)
    tp = int(((treat == 1) & (truth == 1)).sum())
    fp = int(((treat == 1) & (truth == 0)).sum())
    fn = int(((treat == 0) & (truth == 1)).sum())
    tn = int(((treat == 0) & (truth == 0)).sum())
    treated = tp + fp
    return {"policy": name, "TP": tp, "FP": fp, "FN": fn, "TN": tn,
            "treated": treated,
            "accuracy": (tp + tn) / len(truth),
            "precision": tp / treated if treated else float("nan"),
            "recall": tp / (tp + fn) if (tp + fn) else float("nan"),
            "net": tp * BENEFIT - treated * COST}

policy_table = pd.DataFrame([
    price_full("treat nobody", 0),
    price_full("ninety-day rule (incumbent)", scored["lapsed_90d"]),
    price_full("model at 0.50 (software default)",
               (scored["p"] >= 0.50).astype(int)),
    price_full(f"model at {THRESHOLD:.2f} (derived)",
               (scored["p"] >= THRESHOLD).astype(int)),
    price_full("treat everyone (blanket)", 1),
]).set_index("policy")

_br = scored["churned"].mean()
print(f"test base rate {_br:.3f} | "
      f"majority-class accuracy {max(_br, 1 - _br):.3f}")
print(policy_table.round(2).to_string())

# Sensitivity: the threshold inherits its inputs' uncertainty (9.9).
for benefit in (30.0, 40.0, 60.0):
    t = COST / benefit
    treat = (scored["p"] >= t).astype(int)
    tp = int(((treat == 1) & (scored["churned"] == 1)).sum())
    print(f"benefit ${benefit:>5.0f} -> threshold {t:.2f}: "
          f"treated {int(treat.sum()):>5d}   "
          f"net ${tp * benefit - int(treat.sum()) * COST:,.0f}")
```

*Expected output: a header line carrying the test base rate and majority-class accuracy, a five-row policy table with the four confusion counts and five metrics per row, and three sensitivity lines showing how the threshold, the treated-list size, and the expected net move as the benefit estimate varies.*

Input: the sealed test customers and their calibrated scores. Transformation: five treatment vectors run through one pricing function, then the same arithmetic at three benefit estimates. Output: the exhibit the meeting decides on, and the sensitivity the meeting is owed. The precision column carries NaN in the do-nothing row rather than raising a division error, which is the same safe-denominator handling Code 9.2 established and the reason the function is written once rather than five times.

**VERIFICATION CHECK**

Before you run: predict the direction of every movement from the 0.50 row to the 0.30 row. The treated list must grow, recall must rise, precision must fall, and — on data whose economics match the miniature's shape — the expected net must improve, because the program's dear error is the false negative and the lower cut buys fewer of them. Predict also whether the ninety-day rule's net is positive, and whether the blanket's net is positive; the second is arithmetic you can do from the base rate alone, since treating everyone earns 40 × (base rate) and costs 12 per customer.

After you run: re-add the four cells of every row by hand and reconcile each to the test-set count — a matrix that does not sum is a pasted matrix, and pasted matrices are how wrong numbers reach decks. Then write the executive sentence the chapter has been building toward: at the 0.30 threshold, precision below one half means most treated customers will not have churned *by design*, because the program deliberately overspends on the cheap error to avoid the dear one. Finally, read the sensitivity block: if the verdict flips within finance's plausible benefit range, the binding uncertainty is the economics rather than the model.

Investigate if: the derived-threshold row loses money on the test customers after passing the training gate. That is the gap between a cross-validated estimate and a single held-out sample, and the honest report gives both figures with their populations named rather than quoting whichever is friendlier.

### *9.12.9 Lab 9.2, Part F: The Ranked-Targeting Exhibits*

Part F builds the three charts the chapter owns and the decile table beneath them. These are analytic figures rather than presentation graphics; Chapter 12 supplies the visualization theory, and this lab supplies the arithmetic the theory will later dress.

**Code 9.16. Construct the decile table**

```
from sklearn.metrics import roc_curve

scored["decile"] = pd.qcut(
    scored["p"].rank(method="first", ascending=False),
    10, labels=range(1, 11)).astype(int)

base_rate = scored["churned"].mean()
dec = (scored.groupby("decile")["churned"]
       .agg(customers="count", churners="sum", rate="mean"))
dec["lift"] = (dec["rate"] / base_rate).round(2)
dec["cum_capture"] = (dec["churners"].cumsum()
                      / dec["churners"].sum()).round(3)
dec["cum_file"] = (dec["customers"].cumsum()
                   / dec["customers"].sum()).round(3)
print(dec.to_string())
assert np.isclose(dec["cum_capture"].iloc[-1], 1.0)

# The incumbent as one operating point on the same axes.
inc = scored["lapsed_90d"] == 1
inc_file    = inc.mean()
inc_capture = scored.loc[inc, "churned"].sum() / scored["churned"].sum()
```

*Expected output: a decile table with customer counts, churner counts, churn rate, lift, cumulative capture, and cumulative file share, and one assertion that capture reaches 1.0.*

Input: the scored test customers. Transformation: a rank-based decile assignment and three cumulative computations. Output: the arithmetic beneath all three charts. The decile assignment ranks before cutting — `rank(method="first")` breaks ties deterministically — so the ten groups are equal in size regardless of how many customers share a score, which is what makes lift comparable across deciles. The incumbent's file share and capture are computed here too, because the next cell needs them as a single point on two different sets of axes.

**Code 9.17. Plot the ROC, lift, and gains exhibits**

```
fig, axes = plt.subplots(1, 3, figsize=(13.5, 4.2))

fpr_m, tpr_m, _ = roc_curve(scored["churned"], scored["p"])
fpr_r, tpr_r, _ = roc_curve(scored["churned"], p_base)
axes[0].plot(fpr_m, tpr_m, label="finalist")
axes[0].plot(fpr_r, tpr_r, linestyle="-.", label="recency-only baseline")
axes[0].plot([0, 1], [0, 1], linestyle="--", linewidth=1, label="chance")
inc_fpr = ((inc) & (scored["churned"] == 0)).sum() / (
    scored["churned"] == 0).sum()
axes[0].scatter([inc_fpr], [inc_capture], marker="D", zorder=5,
                label="ninety-day rule")
axes[0].set_xlabel("false positive rate"); axes[0].set_ylabel("recall")
axes[0].set_title("ROC"); axes[0].legend(fontsize=8)

axes[1].bar(dec.index, dec["lift"])
axes[1].axhline(1.0, linestyle="--", linewidth=1)
axes[1].set_xlabel("decile (1 = highest)"); axes[1].set_ylabel("lift")
axes[1].set_title("Lift by decile, reference line at 1.0")

axes[2].plot([0] + list(dec["cum_file"]), [0] + list(dec["cum_capture"]),
             marker="o", label="model")
axes[2].plot([0, 1], [0, 1], linestyle="--", linewidth=1,
             label="random targeting")
axes[2].scatter([inc_file], [inc_capture], marker="D", zorder=5,
                label="ninety-day rule")
treated_share = (scored["p"] >= THRESHOLD).mean()
axes[2].axvline(treated_share, linestyle=":", linewidth=1,
                label=f"{THRESHOLD:.2f} cut treats "
                      f"{treated_share:.0%} of the file")
axes[2].set_xlabel("share of file treated")
axes[2].set_ylabel("share of churners captured")
axes[2].set_title("Cumulative gains"); axes[2].legend(fontsize=8)

plt.tight_layout(); plt.show()
print(f"ninety-day rule: treats {inc_file:.1%} of the file, "
      f"captures {inc_capture:.1%} of churners")
```

*Expected output: a three-panel figure carrying the ROC curves with the incumbent marked, lift by decile against its 1.0 reference line, and cumulative gains against the random-targeting diagonal with the economic threshold's treated share marked; then the incumbent's file share and capture printed for the deliverable.*

Input: the decile table and the scored file. Transformation: three plots on shared data. Output: the exhibits the budget conversation runs on. The panels answer three different questions and should be read in that order — whether the ordering is any good at all, how concentrated the churners are at the top of it, and how much of the problem a given depth of file reaches. The ninety-day rule appears on the first and third panels as a single marked point rather than a curve, which is the visual statement of why it does not belong in an AUC column: a fixed policy has one operating point, and a scored file has a curve of them.

**VERIFICATION CHECK**

Before you run: predict the table's shape. Decile 1's churn rate should run well above the base rate, so its lift should sit meaningfully above 2 for a model near this data's ceiling; the bottom deciles should run far below the base rate, which is the half of the exhibit executives forget, because the model is also identifying customers who *do not need* the program. Predict where the gains curve crosses fifty percent capture, and predict whether the ninety-day rule's diamond will sit above or below the model's curve at the same file share.

After you run: read lift with the right expectation. Lift should generally be strongest in the early deciles and weaken as the list deepens; perfect monotonic decline is not guaranteed in a finite test sample, and a non-monotone middle is a finding about where the ordering blurs rather than an error. What would be a failure is a top decile whose lift sits near 1.0. Then write the three sentences the meeting will quote. The capture sentence: the top two deciles contain X percent of all churners — reconcile it against the cum_capture column by hand. The comparison sentence: the ninety-day rule treats Y percent of the file and captures Z percent, placed on the same axes. The budget sentence: at $12 per treatment, funding through decile 2 costs $A and reaches B expected saves worth $C, and the marginal decile stops paying when its incremental capture times $40 falls below its treatment cost.

Investigate if: the economic threshold's vertical line falls far to the right of the deciles the budget can fund. That is the capacity-versus-threshold tension of Section 9.9 arriving in visual form, and the deliverable owes the meeting the priced gap: what the unfunded treatments are expected to return, which is the argument for more budget stated in the only currency budget arguments accept.

### *9.12.10 Lab 9.2, Part G: The Fairness Appendix and the AI Round Trip*

Part G closes the lab with the two audits the deliverable cannot ship without. The first is the fairness appendix proper — not a comparison of average scores, which is what a first draft usually produces, but per-group error rates at the threshold the program will actually use. Section 9.15 argues, and this exhibit shows, that a model can be honest on average and systematically wrong about an identifiable group.

**Code 9.18. Audit group-level errors and score behavior**

```
sc = (scored.join(customers.set_index("customer_id")
                  [["primary_store_id", "home_metro"]])
      .merge(stores[["store_id", "opening_date"]],
             left_on="primary_store_id", right_on="store_id",
             how="left"))

# Newest-store cohort: primary store opened in the 12 months before the
# snapshot. Online-only customers have no store: False, not missing.
sc["new_store"] = sc["opening_date"].ge(
    pd.Timestamp("2025-01-01")).fillna(False)

# A declared lifecycle grouping built only from snapshot-safe columns.
sc["lifecycle"] = np.select(
    [sc["frequency"].eq(1), sc["frequency"].between(2, 3)],
    ["single-purchase", "developing"], default="established")

def group_report(df, by, threshold=THRESHOLD):
    """Treatment rate and both error rates by group, at the threshold the
    program will use. Mean tenure is reported so that any residual
    imbalance in the comparison stays visible. Denominators are checked
    before they divide."""
    out = []
    for key, g in df.groupby(by, dropna=False):
        treat = (g["p"] >= threshold).astype(int)
        y = g["churned"]
        tp = int(((treat == 1) & (y == 1)).sum())
        fp = int(((treat == 1) & (y == 0)).sum())
        fn = int(((treat == 0) & (y == 1)).sum())
        tn = int(((treat == 0) & (y == 0)).sum())
        out.append({
            by: key, "customers": len(g),
            "mean_tenure": g["tenure_days"].mean(),
            "base_rate": y.mean(), "mean_score": g["p"].mean(),
            "treat_rate": float(treat.mean()),
            "precision": tp / (tp + fp) if (tp + fp) else np.nan,
            "recall":    tp / (tp + fn) if (tp + fn) else np.nan,
            "fpr":       fp / (fp + tn) if (fp + tn) else np.nan,
            "fnr":       fn / (fn + tp) if (fn + tp) else np.nan})
    return pd.DataFrame(out).set_index(by).round(3)

for dimension in ["new_store", "home_metro", "lifecycle"]:
    print(f"\n--- by {dimension} ---")
    print(group_report(sc, dimension).to_string())

# The artifact test: repeat the new-store comparison inside a narrow
# tenure band. This RESTRICTS the comparison; it does not match it, so
# the mean_tenure column is what tells you how much imbalance survives.
young = sc[sc["tenure_days"] < 365]
print(f"\n--- by new_store, tenure under one year "
      f"(n = {len(young):,}) ---")
print(group_report(young, "new_store").to_string())
```

*Expected output: three group tables — by new-store status, by metro, and by lifecycle group — each carrying group size, mean tenure, base rate, mean score, treatment rate, precision, recall,*

*false-positive rate, and false-negative rate; then the same new-store comparison recomputed within customers of under one year's tenure.*

Input: the scored test customers joined to the customer and store dimensions. Transformation: three declared groupings and one tenure restriction, each run through identical error arithmetic. Output: the deliverable's fairness appendix. Three construction details carry the section's argument. The new-store flag treats an unmatched store — an online-only customer, whose primary_store_id is the reserved ONLINE value — as False rather than missing, and the choice is declared rather than silent. The lifecycle grouping is built from snapshot-safe behavioral columns rather than from Chapter 6's fitted segments, so the appendix profiles the treated population without importing a learned assignment into a pipeline that never trained on one. And the mean-tenure column exists because the final comparison is a *restriction*, not a match: confining both groups to under a year does not equalize their tenure distributions, and a restricted comparison that still shows one group averaging four months against the other's ten has not removed the artifact it was meant to remove. Report the residual, and say what a genuine match — finer bands, or a matched-pairs design — would take.

**VERIFICATION CHECK**

Before you run: predict the new-store rows using nothing but the store roster and the eligibility rule. Customers whose primary store opened recently are, by construction, recent customers — short tenure, low frequency, thin history — and every one of those features raises the model's churn score by design, so predict a higher mean score, a higher treatment rate, and a higher false-positive rate for the newest cohort. Then predict what the tenure-restricted table will do to the gap, and predict how much tenure imbalance will survive inside the band.

After you run: state the finding in Section 7.2's verbs. The model has not discovered that new-market customers are disloyal; it has discovered that they are new. Read the two mean-tenure figures inside the restricted table before reading anything else: if they still differ materially, the restriction has removed only part of the artifact and the remaining gap cannot be attributed cleanly. Then read the metro and lifecycle tables for any group whose false-positive or false-negative rate departs materially from the aggregate, and write one sentence per departure naming what the program would do differently and what it would cost.

Investigate if: any group's row carries fewer than about fifty customers. Error rates computed on small groups swing wildly, and a fairness appendix that reports a 0.00 false-negative rate on a group of nine has produced noise with a decimal point. Report the group size beside every rate, and say so.

One framing sentence belongs in the deliverable beside this table, and it is a limit rather than a caveat. What Code 9.18 produces is a *minimum diagnostic screen*: evidence that the program's errors were examined across identifiable groups before launch, and a record of what was found. It is not a fairness certification. No single error-parity comparison establishes that a treatment is fair, beneficial, or legally compliant, the parity criteria themselves can conflict with one another, and a program can pass every column in this table and still be a bad idea for the people in it. The screen's honest claim is narrow and worth making: nobody launched without looking.

The second audit is the round trip. Hand an assistant the raw tables and the deliberately loose prompt "build me the best churn model you can — maximize accuracy," and grade the response with Table 8.5's five points followed by Table 9.5's four, in writing: reconstruct the churn definition the assistant invented and compare it to Section 9.3's, naming which levers of Table 9.1 it moved; trace its features across the snapshot, running the recency range check; find the threshold it silently applied; compute the base rate and majority-class baseline it did not report; determine whether any setting was chosen by consulting a test split; look for a reliability exhibit and note whether any dollar figure was computed without one; and re-derive its headline metric from its own confusion matrix. The designed likelihood, engineered by the loose prompt: a label built at the analysis date (S3 and Table 8.2 fail together), a 0.5 threshold nobody chose, a depth or a k selected against test performance, uncalibrated scores feeding an expected-value table, and an accuracy headline whose margin over the majority-class floor was never computed. Your audit memo, filed per Appendix D, is the lab's final deliverable — and the direct rehearsal of the platform-flag verdict Section 9.13 writes.

## 9.13 Marketing Interpretation and Managerial Insight

The lab's outputs are matrices, curves, and a derivation; the twenty-eight-day deadline runs on sentences. This section translates, and — per this guide's standing practice — it does so partly by exhibiting the wrong managerial readings and correcting them, because classification mints misreadings faster than any machinery yet: its numbers sound like everyday words (accurate, precise), its scores sound like verdicts, and both will be spoken in the targeting meeting by people whose only error is trusting the vocabulary.

The first wrong reading arrives holding the platform's documentation: "Their churn flag is 89 percent accurate. Our model's accuracy at the threshold you chose is about the same — and theirs is one click. Why did we build anything?" The correction is not that 89 percent is bad; it is that 89 percent is not yet a number anyone can read. At a base rate near thirty percent, seventy percent accuracy is available for free by treating no one, so the claim's real content is a margin over that floor — a margin that cannot be evaluated without the confusion matrix behind it, the threshold that produced it, and the cost structure that prices the two errors it distributes. And the claim is unauditable besides, because 89 percent accurate *at what definition of churn, at what threshold, on whose customers* are questions the panel's documentation does not answer. Then the reframe the meeting actually needs: accuracy was never the currency. The deliverable's currency is capture against cost — the model's top two deciles reach a stated share of the departing customers at a stated program cost, the platform flag's list (once exported and graded on the same test months) reaches its own share at its own cost, and the two can compete on one gains chart the moment the vendor discloses enough to be graded. The platform verdict is Chapter 8's vendor verdict with this chapter's supplement: the five questions of Table 8.5, plus Table 9.5's four — and the one that ends most such meetings is S3, because a flag that cannot state its threshold's cost assumption is a business policy no one at StyleCraft agreed to.

The second wrong reading is quieter and does more damage per occurrence: "The model says C003914 has a 0.62 churn probability — she's leaving. Why is she still getting the full-price catalog? And C005108 is at 0.08, so we can stop worrying about him." Both sentences collapse a probability into a verdict, and Section 8.13's discipline transfers with the units changed: 0.62 means that among snapshot-similar customers, roughly six in ten went dark — and four in ten did not. The score is not a prophecy about her; it is a rank and a rate, and the program built on it is justified by the *portfolio* — treat enough 0.62s and the arithmetic of Section 9.9 pays reliably — not by any individual call being right. The corrected sentences change the object of confidence exactly as Chapter 8's did: not "she is leaving" but "she belongs to the group most worth treating"; not "he is safe" but "treating his group costs more than it returns." One implication deserves its own sentence in the deliverable, because it defuses the meeting's most predictable ambush: under a 0.30 threshold, the *majority* of treated customers will not have churned even if the model is excellent — precision below one half means most vouchers land on stayers *by design*, because the program deliberately overspends on the cheap error to avoid the dear one. An executive who has not been told this in advance will read the first post-campaign report as proof the model failed; an executive who has been told will read the same report as the cost structure working as priced. Setting that expectation before launch is not spin — it is the threshold decision's second half, and forgetting it has ended more scoring programs than bad models have.

A third misreading is newer to this chapter and specific to its arithmetic: "The model says 0.30 is the cut, so let's use 0.30." That sentence is correct only if the scores near 0.30 mean thirty percent, which is a property the reliability exhibit of Code 9.12 either establishes or refutes. Where it refutes, the threshold is not wrong — the ledger is still 12 against 40 — but the scores are the wrong ruler for it, and the repair belongs in the training data rather than in the meeting. This is the clearest case in the chapter where a number that survives every ranking diagnostic can still price a decision incorrectly, and it is the reason the deliverable carries the calibration exhibit rather than merely mentioning calibration in a footnote.

The deliverable that survives all three misreadings has a fixed anatomy, assembled entirely from lab outputs. Page one is the churn definition per Section 9.3 — the label, its activity basis, its two eligibility conditions, its rejected alternatives, and who each alternative would have added or removed — because every later number is downstream of it. The training-only leaderboard follows, per Section 8.9's fixed rules: three models against the constant-score floor and the recency-only logistic baseline, graded on cross-validated AUC alone, with the base rate printed in the header, the paired fold margins shown, and no dollar figure anywhere on it. Then the selection rule, executed unedited, and the calibration of the selected model inside the training data. Then the threshold derivation as arithmetic — the $12, the $40, the 0.30, the sensitivity at $30 and $60 — stated so the CFO can re-run it, with the out-of-fold reliability exhibit and the economic gate beside it: the calibrated model's cross-validated net value per thousand customers against zero and against the incumbent rule, computed before the test set was opened. A threshold the CFO has re-derived is a threshold the CFO defends thereafter; a threshold applied

to unchecked probabilities is a threshold nobody should defend; and a program that cannot clear its own gate on training evidence is a program the deliverable should decline to fund rather than test its way into. Then the single test-set grade, opened once, on the frozen estimator-plus-calibrator pair. Then the five-policy table at the derived threshold, hand-reconciled, with the precision expectation set in plain words. Then the gains chart with the three policies on one page — model deciles, ninety-day rule, blanket — and the budget sentence the stopping rule produces. The incremental caveat, flagged per Section 7.2 and priced per Section 9.9: the save rate is an assumption this design cannot certify, the experiment that would certify it is named, and the deliverable recommends the program launch *with* a randomly held-out control group, because the cheapest moment to buy causal evidence is before the rollout, not after the doubt. The monitoring plan per Section 8.10, with two classification-specific lines: the base rate itself is monitored, because a drifting base rate silently re-prices the threshold even when the ordering holds; and the calibration is re-checked on each matured cohort, because probabilities drift out of true before rankings do. The permitted-use note per Section 8.15. And the fairness appendix from Lab 9.2 Part G — the per-group error rates, the new-store exhibit, and its tenure-restricted comparison with the residual tenure imbalance reported, framed as a minimum diagnostic screen — which Section 9.15 now takes up on its own terms.

## 9.14 Business Analytics in Practice

This section turns from the fictional case to how churn scoring, retention economics, and classification governance operate in industry — where churn models are among the oldest production analytics in marketing, where the lift chart is a budget instrument, and where the hardest lessons are about what the model made people do. The vignettes below are drawn from published research and from recurring professional patterns rather than from any single named organization.

### *9.14.1 The Telecom Churn Playbook and Two Different Warnings*

The first vignette is the telecom churn playbook, because telecommunications is where churn modeling grew up and where its most instructive surprises were documented. Contractual carriers have modeled churn for three decades: rich behavioral data, a legally crisp label (the cancellation event this chapter's non-contractual setting had to construct by hand), dedicated save desks, and retention offers whose economics are exactly Section 9.9's ledger at industrial scale. The playbook's mature form looks like this chapter's deliverable — scored base, threshold from offer economics, treatment tiers by decile — and it comes with two distinct pieces of scar tissue that students routinely fuse into one and should not.

The first warning is about *targeting*. Ascarza (2018), combining two field experiments with machine-learning targeting rules, found that the customers a churn model ranks riskiest are not necessarily the customers most *responsive* to a retention intervention, so programs aimed by risk rather than by sensitivity to the treatment can misallocate their budgets — and that targeting on estimated response heterogeneity outperformed the standard practice of targeting the highest-risk

customers. In this guide's vocabulary: risk is who is leaving; responsiveness is whom the treatment changes; they are different quantities, and this chapter's observational machinery estimates only the first.

The second warning is stronger and comes from a different study, which is why the two should not be cited interchangeably. Ascarza et al. (2016), in a large-scale field experiment in which some customers received proactive pricing-plan recommendations and some did not, found that the intervention *increased* churn: 10 percent of the treated group churned in the three months following the campaign against 6 percent of the control group, with the authors attributing the effect to lowered inertia about switching and to raised salience of past usage. That is not a modeling error at all — the churn model predicted honestly; the *program* assumed that predicting and preventing are the same problem. Only a held-out control inside the campaign can measure the second, which is Chapter 11's machinery. The practice lesson, planted in Section 9.13's deliverable and repeated here because industry paid heavily to learn it: launch scored programs with controls from day one, and treat "intervention is not prediction" as a design constraint rather than a footnote.

### *9.14.2 The Lift Chart as a Budget Instrument*

The second vignette is the lift chart as a budget instrument, because the exhibit students meet as a model-evaluation chart functions in industry as a finance document. At subscription and retail firms with mature retention programs, the annual retention budget is negotiated *off the gains curve*: the analytics team presents cumulative capture by decile, finance attaches the treatment cost and save economics — the same ledger as Section 9.9, at line-item scale — and the budget conversation becomes a stopping-rule conversation: fund deciles while the marginal decile's expected saved margin clears its treatment cost, stop where the curve's slope says to stop. Teams that operate this way report the cultural shift this chapter's Concept box predicted: the argument stops being "is the model good" — a question finance cannot engage — and becomes "where does the curve stop paying," a question finance is better at than the analysts.

The recurring failure mode is equally instructive: programs that set the treated-list size first (a round number, a platform tier limit, last year's list) and never reconcile it against the curve, so that the budget quietly funds deciles the arithmetic abandoned — the capacity-versus-threshold tension of Section 9.9, resolved by inertia instead of by the priced argument. A second failure is subtler and belongs to this chapter's calibration thread: teams that read the gains curve correctly and then compute the program's expected return from raw model scores that nobody checked for calibration, producing a budget denominated in probabilities the model does not actually print. The lesson is the closing thread's preview: the decile table is where analytics and finance speak the same language, and the analyst who brings the curve — and the reliability exhibit that entitles its dollar figures — to the budget meeting sets the agenda.

### *9.14.3 When a Marketing Score Crosses a Regulatory Line*

The third vignette is the regulatory boundary, because propensity machinery does not stay in marketing. The same scored file that ranks customers for retention vouchers can rank them for credit-like treatments — preapproved financing offers, deposit requirements, payment-plan eligibility, service-tier routing — and the moment it does, it leaves the domain of marketing judgment and enters one governed by statute, supervision, and litigation.

The practice lesson here is a boundary discipline rather than a legal conclusion, and the reason is that the legal ground genuinely moves. In the United States, the federal treatment of disparate impact in credit was substantially revised in 2026: the Consumer Financial Protection Bureau published a final rule amending Regulation B, effective July 21, 2026, stating that the Equal Credit Opportunity Act does not authorize disparate-impact liability and removing the effects test from the regulation (Consumer Financial Protection Bureau, 2026). Commentators note that disparate-impact theories may still operate under other federal statutes and under state fair-lending law, and litigation over the rule's scope was active at the time of writing. This chapter therefore declines to state a general rule and states an obligation instead: credit-adjacent uses of customer scores can trigger federal and state fair-lending, discrimination, adverse-action, explanation, and consumer-protection requirements; the governing rules are jurisdiction- and use-specific and change; and they should be reviewed with legal and compliance specialists before deployment rather than reasoned out from a textbook paragraph.

What does not change is the mechanism the analyst is uniquely positioned to see, and it is the one Barocas and Selbst (2016) mapped and Section 6.16 previewed for segmentation: neutral-looking features such as geography, tenure, or shopping channel can carry protected characteristics by correlation, so a model that never sees a protected attribute can still distribute its errors unequally across groups defined by one. Mature organizations institutionalize the boundary exactly the way this guide's permitted-use note does: the score's approved uses are written down, credit-adjacent repurposing triggers legal review and formal fairness testing across groups, and the analyst of record is the person expected to notice when a marketing score is drifting toward a regulated decision. The lesson: the fairness screen of Lab 9.2 Part G is not an academic garnish — it is the junior version of an audit that, in adjacent industries, is performed under subpoena.

### *9.14.4 In Your First Analyst Job*

In your first analyst job, these vignettes compress into one expectation, and it is this chapter's closing thread: threshold-setting is where the analyst meets the CFO. The model will be a commodity — every platform ships one, every assistant drafts one — and the meetings that matter will be about the numbers around it: what the label means, what the two errors cost, whether the probabilities mean what they say, where the curve stops paying, what the program may and may not be used for, and whether anyone built the control group that can prove the treatment works. Every one of those is a conversation this chapter equipped, and none of them is a modeling conversation. The analysts who advance are the ones who can hold the room through

the cost arithmetic — because the threshold is the one number in the pipeline that the CFO, once shown, will insist on owning, and the analyst who taught the CFO to own it becomes the person the CFO calls before every scored program thereafter.

## 9.15 Ethics, Fairness Across Segments and the Loops Models Close

The Business Analytics in Practice section ended at a legal boundary; this section examines the ethical ground on both sides of it, extending the guide's running discussions — data use (Section 1.13), problem framing (Section 2.12), measurement design (Section 3.13), cleaning as editorial power (Section 4.14), honest summarization (Section 5.14), differential treatment of segments (Section 6.16), causal language (Section 7.15), and acting on predictions about people (Section 8.15) — to what classification adds: verdicts about individuals, sorted into groups, at scale, by machinery whose errors are invisible one customer at a time. Two obligations are new here, and the dataset was designed to make the first one unmissable.

The first obligation is fairness across segments, and its instrument is one this chapter already built: the confusion matrix, computed *by group*. A model's aggregate metrics — the leaderboard AUC, the overall precision and recall — are averages across the file, and averages hide people; that was Chapter 5's ethics, and it returns here with teeth, because a classifier can post excellent aggregate numbers while distributing its errors unequally across identifiable groups: flagging one group's loyal customers as churners at twice the rate it does another's, or missing one group's departing customers systematically. The new-store exhibit of Lab 9.2 Part G is the designed demonstration — the second strike of the dataset's ethics trap, and the sharper one. The first strike, in Chapter 6, was about deliberate differential treatment of known segments; this strike is about *accidental* differential judgment of a group nobody defined: customers of the newest stores, whose short tenure and thin history are artifacts of their stores' opening dates and StyleCraft's own under-marketing during ramp-up, and whom a naive churn model therefore scores as high-risk for reasons that describe the company's history, not the customers' intentions. A retention program acting on the raw scores would flood the newest markets with discount vouchers — training precisely the wrong price expectations in the expansion's youngest relationships — and, worse, the same scores repurposed upstream ("the new markets show terrible predicted retention") would argue for abandoning stores whose only sin is being new. The dataset's designers built the trap so that the correction is achievable with this guide's own tools: the tenure-restricted comparison of Code 9.18, the store-cohort controls the exercises rehearse, and the standing rule that the deliverable's fairness appendix reports error rates by segment, by metro, and by store cohort before any scored program launches. The disparate-impact frame of Barocas and Selbst (2016), cited since Section 1.13, names the general mechanism: models trained on data shaped by past decisions re-encode those decisions as predictions, and the analyst who does not look for the re-encoding has, by default, endorsed it.

The second obligation has no descriptive-analytics ancestor, because it belongs to models that act: the feedback loop, in which the model helps cause the outcome it predicts, and its own future training data records the episode as vindication. The mechanisms are concrete and none

requires malice. A customer flagged as likely to churn is routed to reduced marketing investment — why spend on the leaving? — and, starved of contact, drifts away: the prediction manufactured its own confirmation, and next year's training data files her under "correctly predicted churner," making next year's model more confident in the judgment that caused the loss (O'Neil, 2016). The retention offer itself loops in the other direction: treated customers who stay are recorded as stayers, so the treatment contaminates the label, and a model retrained naively on post-program data learns that high-risk profiles stay — because the program kept saving them — degrading exactly where the program worked; the repair, at concept level, is to train on untreated customers or on the control group the Section 9.13 deliverable insisted on, and the full machinery is one more argument for randomized holdouts as standing infrastructure rather than occasional experiments. Even the churn-increasing intervention of Section 9.14 is a feedback loop in miniature: the model's list determined who received the contact that changed the outcome the model was predicting. The general statement deserves the chapter's last box, because it is the deepest way this chapter's subject differs from every one before it: a descriptive summary leaves the world as it found it; a deployed classifier is an *intervention wearing an observation's clothes*, and the analyst who deploys one inherits a monitoring obligation — not just "is the model still accurate" (Section 8.10's decay) but "is the model changing the population it scores," which no accuracy metric detects.

**CONCEPT**

**The Second Strike, and the Loop**

The guide's running ethics thread has moved from data (what may be collected), through framing and measurement (what gets asked and counted), to summaries and segments (who gets averaged and who gets grouped), to predictions (who gets acted on). Classification completes the arc: verdicts, at scale, that can differ unfairly across groups and can cause what they claim to foresee.

The two standing instruments this chapter adds to the deliverable — the fairness screen (error rates by group, with artifact checks like the tenure-restricted new-store comparison) and the control group (the only reliable detector of a model's own causal footprint) — cost little and are, in this guide's judgment, non-negotiable for any scored program that touches customers. Neither is a certification: passing an error-parity screen does not establish that a program is fair, and a control group measures the treatment's effect rather than its ethics. What they establish is that someone looked, on the record, before launch. The analyst of record signs the scores; from this chapter forward, the signature covers not only what the model predicts but what the program does to the people predicted about.

Source: Course concept developed for this guide, informed by Barocas and Selbst (2016) and O'Neil (2016).

## 9.16 Chapter Summary

This chapter completed the predictive machinery Part II has been assembling, by carrying it across the line from how much to whether. The frame, the label wall, the leakage audit, the training-only selection protocol, the sealed test set, and the fixed leaderboard all transferred from

Chapter 8 intact; what changed was the geometry of being wrong. A binary decision fails in two directions with two prices, and the chapter's whole architecture followed from refusing to average those prices: the confusion matrix kept the four outcomes separate and named them in program vocabulary, with its cost column priced against the program's own ledger rather than against the full margin of a lost customer; precision and recall gave the two stakeholders their two questions; accuracy was demoted to a number reportable only beside its base rate and the do-nothing floor, with the margin between them computed rather than implied; AUC graded the ordering the scorers produce, on its full zero-to-one range, with ties counted as half and with its silences stated — and was assigned the job it can do, choosing a model family, rather than the job it cannot, pricing a program; and calibration was promoted from a footnote to a gate, because the chapter's destination multiplies probabilities by dollars and a well-ordered score can still be the wrong ruler.

The chapter's destination redeemed a seven-chapter-old promise: the asymmetric cost of being wrong, introduced in Chapter 2 as interpretive discipline and previewed in Chapter 8 as expected-value framing, became the decision threshold — twelve dollars against forty, a cut at 0.30 instead of the default's 0.5, derived as arithmetic a CFO can re-run, shown with the sensitivity that reveals whether the model or the economics is the binding uncertainty, and applied only after a reliability exhibit established that the probabilities could bear it. Around that center, the chapter built the working instruments of scored marketing: the churn label as a measurement decision with a policy inside it, defined from purchase activity rather than from net revenue and carrying both of Chapter 8's eligibility conditions; logistic regression reading risk in odds, with trees and neighbors as structurally different second opinions whose flexibility was chosen inside the training folds rather than declared; and the behavioral propensity exhibits — deciles, lift, gains — that turn a scored file into a budget argument with a stopping rule, carefully distinguished from the treatment-assignment propensity score Chapter 11 will need.

The AI section extended the five-point audit with the classification supplement — base rate and floor, matrix arithmetic by hand, threshold provenance, calibration and selection hygiene — and the labs ran the whole discipline twice: once at hand scale on ten customers where three policies tied at exactly 0.70 accuracy while netting nothing, thirty-two dollars, and forty-eight dollars — and where the incumbent rule, tied with none of them, lost eight — and once at full scale across eighteen numbered cells, where flexibility was chosen by a one-standard-error rule, the family was chosen on ranking evidence alone, the selected model was calibrated inside the training data and made to clear an economic gate before any money was discussed, and the deliberate failures were manufactured and caught without ever spending the test set on a diagnostic. The interpretation, practice, and ethics sections carried the scores into the rooms that matter: the platform flag met the supplement's questions, the probability was defended from becoming a verdict, the threshold was defended from being applied to unchecked scores, industry's own scar tissue was read at market prices — with the churn tournament's actual finding restored, the two Ascarza results separated, and the 2026 fair-lending change treated as a moving legal boundary rather than a settled rule — and the deliverable acquired its two standing

appendices: the fairness screen that catches a model judging customers for their stores' age, and the control group that catches a model causing what it predicts.

Looking ahead, the machinery now predicts individuals — their spend, their departure — from a snapshot of their history. The next family of marketing questions abandons the individual for the aggregate and the snapshot for the flow of time itself: what will revenue be in December, how will the new stores' ramp reshape the season, what should the buy plan assume about demand that has not happened yet? Time changes the rules more deeply than the target type did. Observations are no longer exchangeable customers but ordered days, the random splits this Part has relied on would let the future leak into the past wholesale, and the baselines with teeth stop being spreadsheets and become the calendar's own rhymes — tomorrow resembles today, December resembles last December. The next chapter takes up forecasting demand, sales, and campaign performance: trend, seasonality, and the discipline of predicting a future the model must be graded against honestly — including the one event StyleCraft's own history planted in the series, the expansion's inflection, which any forecast worth funding has to see coming.

## 9.17 Exercises for Practice and Homework

The following exercises practice the chapter's main habits: define the label in writing before touching data, carry the base rate into every exhibit, verify confusion-matrix arithmetic by hand, choose model settings inside the training folds, check calibration before computing dollars, derive thresholds from stated costs rather than inheriting them, grade rankings with lift and gains against real incumbents, audit AI-built classifiers with the supplement, and check every scored program's behavior across customer groups. They are organized into three groups. Core chapter practice is the required path and should be completed by every student; it holds the two homework submissions from which your instructor will assign a subset, and both serve as direct preparation for Project #1, whose brief is released with this session and whose rubric appears in Appendix F. In-class activities are prepared for discussion rather than submitted. Extensions are optional. Each exercise also carries its assignment label so that instructors can assign selectively.

### *9.17.1 Core Chapter Practice*

**Exercise 9.1 Concept Check (Required Practice)**

Answer each in two or three sentences, in your own words.

- Distinguish classification from class-probability estimation, and explain why this guide insists the threshold that converts one into the other is not a model property.
- Explain why churn must be *defined* rather than observed in a non-contractual business, and name the three definitional levers of Table 9.1 with one consequence each.
- The chapter builds the churn label from counted purchase orders rather than from zero net revenue. Give two concrete ways the two definitions can disagree, and state which one the deliverable should report.

State both conditions of this chapter's eligibility rule, and explain what breaks in the feature table if the second condition is dropped.

A logistic coefficient on discount_share is 1.1, exponentiating to about 3.0. Write the honest plain-language sentence, including the ceteris paribus clause and a verb that survives Section 7.2.

Why does k-NN require the scaling discipline of Section 6.7 when logistic regression does not require it for validity?

A churn model posts 88 percent accuracy on a file whose base rate is 9 percent. Compute the majority-class baseline, state the margin, and state what the comparison does and does not establish.

State precision's question and recall's question in Comeback Edit vocabulary, name the stakeholder who owns each, and explain why the threshold moves them in opposite directions.

AUC is 0.77. Write the one-sentence probabilistic interpretation including the treatment of ties, state the full range AUC can occupy and what a value below 0.5 would mean, and name two things this number does not certify.

Define calibration, explain how it differs from discrimination, and state precisely which step of the Section 9.9 derivation fails when scores are well ordered but systematically too high.

An analyst inherits a confusion matrix cut at 0.5 and is told "that's just the standard." State the cost assumption the default embodies, name the two further conditions that ride inside the equal-cost reading, and show with this chapter's ledger what threshold the program's own economics imply.

Distinguish the behavioral propensity score of this chapter from the treatment-assignment propensity score of causal inference, and give one sentence on why confusing them would matter in Chapter 11.

**Exercise 9.2 Confusion-Matrix and Policy Arithmetic (Required Practice)**

Using only the ten-customer miniature of Lab 9.1 Part A (no code): rebuild both confusion matrices — thresholds 0.50 and 0.30 — customer by customer, and verify every figure in the Verification Check: the cell counts, both accuracies, both precisions, both recalls, and the F1 at each threshold (approximately 0.57 and 0.67; show the harmonic-mean arithmetic). Then rebuild the five-policy economics table — treat nobody, the 0.50 cut, the 0.30 cut, the ninety-day rule, and the blanket — and verify the nets of $0, +$32, +$48, −$8, and $0. State explicitly what your table does with precision in the do-nothing row and why that is not zero. Identify the single customer whose reclassification between the two thresholds contributes most to the net improvement, and write one sentence on what her profile says about which customers live near a threshold.

**Exercise 9.3 Derive the Threshold (Required Practice)**

For each program below, derive the economic threshold from the stated costs, state the treatment rule in one sentence, and note anything the arithmetic reveals about the program's design. Close each with one sentence on what calibration evidence you would require before applying your threshold to a model's raw scores.

- A win-back email costs $3 per treated customer once redemptions and margin give-up are blended, against a $30 expected benefit per treated true churner. Derive the threshold and explain what a very low threshold implies about who should receive the email — and what Section 9.14's first vignette warns even "nearly free" treatments can cost.
- A high-touch save program — a personal stylist call plus a $40 credit — costs $55 per treated customer, against an expected benefit of $110 per treated true churner. Derive the threshold and state, using the gains-chart logic of Section 9.10, why this program and the email program should not share a treated list.
- A program's benefit estimate is uncertain: finance brackets it between $24 and $72, against a $12 treatment cost. Derive the threshold at both ends, and write two sentences on what the range means for the deliverable, per Section 9.9's sensitivity discipline.

**Exercise 9.4 Spot the Failure (Required Practice)**

Each scenario below contains at least one failure from this chapter (or none). Name it, cite the section or table row, and state the repair.

- A churn model's features include days_since_last_purchase computed at the June 30, 2026 analysis date, for a label whose outcome window began January 1, 2026. Its test AUC is 0.96.
- A team fits logistic regression, a tree, and a k-NN, compares all three on test AUC, then tries three tree depths and keeps the one with the best test AUC. Everything else in the pipeline is clean.
- An assistant, told the classes are imbalanced, oversamples churners until the training data is 50/50, then computes an expected-value table at the 0.30 threshold from the resulting scores.
- A deck reports precision of 0.81 and recall of 0.74 for the same model. A footnote reveals the two figures were achieved at different thresholds.
- A team evaluates its churn model with a stratified 80/20 split, selects model family and settings by stratified cross-validation inside the training customers, reports test AUC beside the majority-class baseline and the incumbent rule, and prints an out-of-fold reliability table beside its threshold derivation.
- A churn model's AUC is 0.79 and its scores near the economic threshold run about fifteen percentage points above the observed churn rate in those bins. The team applies the 0.30 cut anyway, on the grounds that AUC is strong.

A retention program treats the top decile, and the following year's retrained model shows the old top decile's profiles now churn at nearly the base rate. The team concludes the original model was wrong.

A model's aggregate recall is 0.72, but recall computed within the resort-metro cohort is 0.31. The deck reports only the aggregate.

**Exercise 9.5 The Full Churn Deliverable (Homework Submission)**

Complete Labs 9.1 and 9.2 on the certified files and assemble the targeting deliverable per Section 9.13: the label definition page with its activity basis, both eligibility conditions, and the rejected alternatives; the training-only leaderboard with three models, the constant-score floor, and the continuous-recency baseline on cross-validated AUC and expected net value, with the base rate stated and the paired fold margins shown; the declared selection rule and the single sealed-test grade; the threshold derivation with its sensitivity table and its out-of-fold reliability exhibit; the hand-reconciled five-policy table at the derived threshold, including the ninety-day rule and the blanket, with the precision-expectation paragraph for the executive reader; the ROC, lift, and gains figures with the incumbent marked as an operating point; the budget sentence the stopping rule produces; the incremental caveat with the control-group recommendation; the monitoring plan including base-rate and calibration monitoring; the permitted-use note; and the fairness appendix with per-group error rates, the new-store exhibit, its tenure-restricted comparison with the residual tenure imbalance reported, and the sentence framing it as a minimum diagnostic screen rather than a certification. Submissions are graded on the threshold derivation, the calibration evidence, and the fairness appendix as heavily as on the models — and this deliverable's structure is the template for Project #1's predictive track (Appendix F).

**Exercise 9.6 AI Classification Audit (Homework Submission)**

Give an AI assistant the raw StyleCraft tables and this deliberately loose prompt: "Build me the best churn model you can — maximize accuracy." Audit the response in writing with Table 8.5's five points followed by Table 9.5's four: reconstruct the churn definition the assistant invented and compare it against Section 9.3's, naming which levers of Table 9.1 it moved and whether it built the label from activity or from revenue; trace every feature across the snapshot, running the recency range check; identify the threshold it applied and whether it disclosed applying one; compute the base rate and majority-class baseline it omitted and state the margin; determine whether any model family or setting was chosen by consulting a test split; note whether any calibration evidence accompanies any dollar figure; and re-derive its headline metrics from its own confusion matrix, reconciling the cell counts by hand. Then re-prompt with the full frame per the AI in Practice box and compare the two responses' leaderboards. Document both exchanges per the AI-use documentation template in Appendix D, and conclude with two sentences on which supplement point caught the most serious problem.

### *9.17.2 In-Class Activities*

**Exercise 9.7 The Fairness Slice (In-Class Discussion)**

Your team will prepare the fairness appendix conversation for the Comeback Edit launch meeting. Prepare: the new-store exhibit from Code 9.18, with the tenure-restricted comparison and a one-paragraph explanation, in Section 7.2's verbs, of why the raw score gap is largely an artifact; per-group error rates (by metro and by lifecycle group) at the 0.30 threshold, identifying any group whose false-positive or false-negative rate departs materially from the aggregate and reporting every group's size beside its rates; a recommendation on what the program should do differently, if anything, for new-store customers — with the trade-offs stated, since both over-treating and excluding them have costs; and a position on the question the meeting will actually ask: "the model is accurate overall — why are we complicating the launch?" Half the class prepares that skeptical position seriously; the strongest version of the skeptic's case is what the fairness appendix must be built to survive. Close by stating, in one sentence, what your screen does *not* establish.

**Exercise 9.8 Find the Flaw in the Scored Program (In-Class Discussion)**

Each scenario below contains at least one flaw from this chapter. Name it, cite the section, and state the repair.

- A retention program celebrates its first quarter: 60 percent of treated customers stayed. No control group was held out, and the team cannot say what fraction would have stayed untreated.
- A vendor's churn module is adopted because its 0.94 AUC beat the in-house model's 0.77. The vendor's features are proprietary, and its evaluation data cannot be inspected.
- The Comeback Edit list is cut at the budget's 1,600-customer capacity. Nobody has computed where the economic threshold falls, so nobody knows whether the budget is too small, too large, or right.
- A team tunes its classifier to maximize F1, presents the result as "optimal," and cannot say what the program's false-positive and false-negative costs are.
- A deck reports that the model's top decile has a lift of 1.05 and recommends funding the top two deciles because "the model beats random."
- After a successful year, the churn scores are adopted by the finance team to set deposit requirements for a new payment-plan product. No new review occurs, because "the model is already validated."
- A model predicts churn for the newest cohort at twice the file's rate. The expansion review deck cites this as evidence the new markets are failing. State the two distinct analyses from this chapter that the deck skipped.

### *9.17.3 Extensions*

**Exercise 9.9 Customer-Specific Thresholds (Optional)**

Section 9.9 assumed one benefit figure for every customer, and Chapter 5's concentration findings guarantee that assumption is false. Using the scored test file, replace the single $160 retained-margin figure with a customer-specific estimate — the customer's own feature-window revenue_per_month projected across the planning horizon is a defensible starting point, and you should defend whatever you choose — then derive a customer-specific threshold for each customer and recompute the five-policy table with the personalized rule added as a sixth row. Report three things: how the treated-list size changes, how the expected net changes, and which customers enter or leave the list. Then write two paragraphs on what this refinement costs. One should address measurement (the personalized benefit is itself an estimate with its own error, and the threshold now inherits two uncertainties instead of one); the other should address governance (a program in which different customers face different treatment thresholds is harder to explain, harder to audit for the group effects of Section 9.15, and harder to defend if the score is ever repurposed).

## 9.18 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.

*Accuracy.* The share of evaluated cases whose predicted class matches the actual class; under class imbalance, dominated by the majority class and reportable only beside the base rate and the majority-class baseline, with the margin between them computed (adapted from Provost & Fawcett, 2013).

*AUC (area under the ROC curve).* A threshold-free summary of ranking skill ranging from 0 to 1, equal to the probability that a randomly chosen positive case receives a higher score than a randomly chosen negative one, with ties counted as half. A value of 0.5 indicates chance-level ordering and values below 0.5 indicate a systematically reversed or worse-than-chance ordering; the measure is silent about calibration and costs (adapted from Fawcett, 2006; Google for Developers, 2026).

*Base rate.* The share of the positive class in the evaluated population; the context number without which no classification metric is legible, and itself a constructed quantity under a constructed label (adapted from Provost & Fawcett, 2013).

*Behavioral propensity score.* A model-estimated probability of a defined customer behavior, used to rank customers for differential treatment; the packaged form in which this chapter's machinery appears throughout the marketing stack, and distinct from the treatment-assignment propensity score of causal inference (adapted from Provost & Fawcett, 2013).

*Calibration.* The property of a probability scorer whose predicted probabilities match observed event frequencies, so that among cases scored near p about a share p are positive; distinct from discrimination, assessed with a reliability exhibit of mean predicted probability

against observed rate by score bin, and required before an economically derived threshold may be applied to raw scores (adapted from scikit-learn developers, 2026a).

*Churn label.* The operationalized record of churn for a historical case: a defined activity — here, at least one completed purchase order — failing to occur within a defined outcome window after the snapshot, under stated eligibility rules; a measurement decision whose definitional levers change the model and the treated population (adapted from Neslin et al., 2006; Provost & Fawcett, 2013).

*Class imbalance.* The condition in which one class heavily outnumbers the other, under which accuracy is dominated by the majority class and rank-based or cost-based evaluation is required (adapted from He & Garcia, 2009).

*Class-probability estimation.* The prediction, for each case, of the probability of the positive class — a score that ranks cases and defers verdicts to a separately chosen threshold (adapted from Provost & Fawcett, 2013).

*Classification.* The prediction of a categorical target by assigning each case to a class; in this guide, produced by thresholding estimated probabilities rather than emitted directly by the model (adapted from Provost & Fawcett, 2013).

*Classification tree.* A model that predicts by routing cases through a learned sequence of yes/no feature questions to leaves, where predicted probabilities are the leaves' training-class shares; grown by recursive splitting and governed by depth or leaf-size limits chosen inside the training data (adapted from Breiman et al., 1984).

*Confusion matrix.* The cross-tabulation of actual against predicted classes at a stated threshold, partitioning cases into true positives, false positives, false negatives, and true negatives; the arithmetic source of every threshold-dependent metric (adapted from Provost & Fawcett, 2013; Fawcett, 2006).

*Decision threshold.* The probability above which a scored case is treated as positive; a policy choice allocating errors between false positives and false negatives, derived from the two errors' relative costs rather than from the software default of 0.5, and valid only where the scores are sufficiently calibrated (adapted from Provost & Fawcett, 2013; scikit-learn developers, 2026b).

*F1 score.* The harmonic mean of precision and recall, dragged toward the smaller of the two; a single-number summary that weights the two errors equally and is therefore provisional wherever costs are asymmetric (adapted from Provost & Fawcett, 2013).

*False negative.* A positive case predicted negative — here, a departing customer no one tried to save; priced as the missed expected incremental save rather than as the customer's full future margin (adapted from Provost & Fawcett, 2013).

*False positive.* A negative case predicted positive — here, a retention offer spent on a customer who was staying; priced as the blended treatment cost, and bought deliberately by low thresholds (adapted from Provost & Fawcett, 2013).

*Gains chart.* The cumulative share of all positive cases captured as treatment extends down the score-ranked file, plotted against the share of the file treated; read against the random-

targeting diagonal and used with treatment economics as a budget stopping rule (adapted from Provost & Fawcett, 2013; Neslin et al., 2006).

*k-nearest neighbors (k-NN).* A model that predicts a case's class probability as the positive share among its k closest training cases in scaled feature space; coefficient-free, with flexibility governed by k and k chosen by cross-validation inside the training data (adapted from Cover & Hart, 1967).

*Lift.* The ratio of a targeted group's positive rate to the base rate — how many times richer in the target class a model-selected slice is than a random one; the ranked-targeting metric with its baseline built in (adapted from Provost & Fawcett, 2013).

*Log-odds.* The natural logarithm of the odds; the unbounded, symmetric rescaling of probability on which logistic regression's linear equation operates (adapted from Hosmer et al., 2013).

*Logistic function.* The S-shaped curve that converts any log-odds value into a probability between 0 and 1 — flat near the extremes, steepest at even odds (adapted from Hosmer et al., 2013).

*Logistic regression.* The model that expresses the log-odds of a binary target as a linear function of features and recovers probabilities through the logistic function; its exponentiated coefficients are read as multiplicative effects on odds, ceteris paribus (adapted from Hosmer et al., 2013; James et al., 2021).

*Odds.* The ratio of an event's probability to its complement's — $p \div (1 - p)$; the currency in which logistic coefficients speak after exponentiation (adapted from Hosmer et al., 2013).

*Precision.* The share of predicted positives that are actually positive — the purity of the treated list, and finance's question; undefined rather than zero when nothing is treated (adapted from Provost & Fawcett, 2013).

*Recall (true positive rate).* The share of actual positives predicted positive — the coverage of the at-risk class, and the churn-number owner's question (adapted from Provost & Fawcett, 2013; Fawcett, 2006).

*ROC curve.* The plot of true positive rate against false positive rate across all thresholds, tracing a scorer's ranking trade-off independent of any single cut (adapted from Fawcett, 2006).

*True negative.* A negative case predicted negative — the quiet majority correctly left untreated, and the cell that inflates accuracy under imbalance (adapted from Provost & Fawcett, 2013).

*True positive.* A positive case predicted positive — here, a save opportunity reached (adapted from Provost & Fawcett, 2013).

## 9.19 Further Readings

Students who want additional background may begin with the following readings. The business-first treatments are listed first, methods second, software and consequences last.

Provost and Fawcett (2013), especially the chapters on model evaluation, expected value, and ranking (lift and profit curves), for the business-first development of nearly everything this chapter built — the closest published relative of its approach, as it was for Chapter 8.

Neslin et al. (2006) for the churn-modeling tournament that graded forty-five submissions on common validation data and found that modeling approaches differ materially in predictive and economic performance — the empirical case for this chapter's leaderboard discipline, and a useful corrective to the folk claim that method choice does not matter.

Fawcett (2006) for the standard accessible introduction to ROC analysis — the extended companion to Section 9.8, with the practitioner's cautions included.

Ascarza (2018) and Ascarza et al. (2016), read as a pair: the first demonstrates that targeting by churn risk can misallocate retention because risk is not responsiveness, and the second demonstrates, in a field experiment, that a proactive retention campaign can raise churn outright. Together they are the cleanest bridge from this chapter to Chapter 11.

James et al. (2021), Chapter 4, for the standard development of logistic regression, k-NN, and classification evaluation, one level of formality above this chapter — and, for the implementation of exactly these ideas, the three pages of official scikit-learn documentation these labs are built on (scikit-learn developers, 2026a, 2026b, 2026c): probability calibration, post-hoc threshold tuning, and the cross-validation guidance that keeps development inside the training folds.

O'Neil (2016) for the general-audience treatment of feedback loops and scaled algorithmic harm — the accessible companion to Section 9.15's second obligation, read alongside Barocas and Selbst (2016) for the scholarly treatment of the disparate-impact mechanism itself.

## 9.20 References


Ascarza, E. (2018). Retention futility: Targeting high-risk customers might be ineffective. *Journal of Marketing Research, 55*(1), 80–98. https://doi.org/10.1509/jmr.16.0163

Ascarza, E., Iyengar, R., & Schleicher, M. (2016). The perils of proactive churn prevention using plan recommendations: Evidence from a field experiment. *Journal of Marketing Research, 53*(1), 46–60. https://doi.org/10.1509/jmr.13.0483

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

Breiman, L., Friedman, J. H., Olshen, R. A., & Stone, C. J. (1984). *Classification and regression trees*. Wadsworth.

Consumer Financial Protection Bureau. (2026). *Equal Credit Opportunity Act (Regulation B)*, 12 C.F.R. pt. 1002. https://www.consumerfinance.gov/rules-policy/regulations/1002/

Cover, T. M., & Hart, P. E. (1967). Nearest neighbor pattern classification. *IEEE Transactions on Information Theory, 13*(1), 21–27. https://doi.org/10.1109/TIT.1967.1053964

Fawcett, T. (2006). An introduction to ROC analysis. *Pattern Recognition Letters, 27*(8), 861–874. https://doi.org/10.1016/j.patrec.2005.10.010

Google for Developers. (2026). *Classification: ROC and AUC* [Machine Learning Crash Course]. https://developers.google.com/machine-learning/crash-course/classification/roc-and-auc

He, H., & Garcia, E. A. (2009). Learning from imbalanced data. *IEEE Transactions on Knowledge and Data Engineering, 21*(9), 1263–1284. https://doi.org/10.1109/TKDE.2008.239

Hosmer, D. W., Lemeshow, S., & Sturdivant, R. X. (2013). *Applied logistic regression* (3rd ed.). Wiley. https://doi.org/10.1002/9781118548387

James, G., Witten, D., Hastie, T., & Tibshirani, R. (2021). *An introduction to statistical learning: With applications in R* (2nd ed.). Springer. https://doi.org/10.1007/978-1-0716-1418-1

Neslin, S. A., Gupta, S., Kamakura, W., Lu, J., & Mason, C. H. (2006). Defection detection: Measuring and understanding the predictive accuracy of customer churn models. *Journal of Marketing Research, 43*(2), 204–211. https://doi.org/10.1509/jmkr.43.2.204

O'Neil, C. (2016). *Weapons of math destruction: How big data increases inequality and threatens democracy*. Crown.

Pedregosa, F., Varoquaux, G., Gramfort, A., Michel, V., Thirion, B., Grisel, O., Blondel, M., Prettenhofer, P., Weiss, R., Dubourg, V., Vanderplas, J., Passos, A., Cournapeau, D., Brucher, M., Perrot, M., & Duchesnay, É. (2011). Scikit-learn: Machine learning in Python. *Journal of Machine Learning Research, 12*, 2825–2830.

Provost, F., & Fawcett, T. (2013). *Data science for business: What you need to know about data mining and data-analytic thinking*. O'Reilly Media.

scikit-learn developers. (2026a). *Probability calibration* [Software documentation]. https://scikit-learn.org/stable/modules/calibration.html

scikit-learn developers. (2026b). *Post-hoc tuning the cut-off point of decision function* [Software documentation]. https://scikit-learn.org/stable/auto_examples/model_selection/plot_tuned_decision_threshold.html

scikit-learn developers. (2026c). *Cross-validation: Evaluating estimator performance* [Software documentation]. https://scikit-learn.org/stable/modules/cross_validation.html