# L03: Alternative Credit Scoring & ML Pipelines

## Extended Slides – BSc Digital Finance Course

Digital Finance

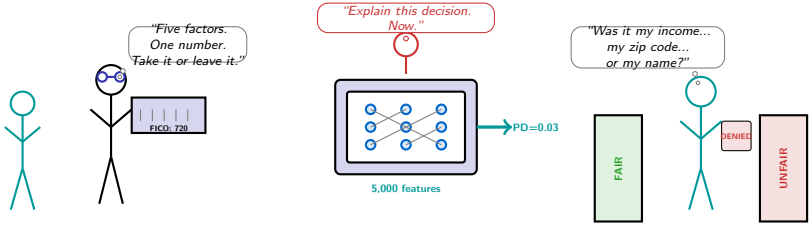
# What Will You Be Able to Do After This Lecture?

1. Derive the logistic regression model from first principles — log-odds, MLE objective, and gradient update — and explain why it remains the regulatory baseline for credit scoring
2. Engineer discriminative features from raw credit data, quantify feature importance via information gain, and detect multicollinearity using correlation clustering
3. Compare classification models (logistic regression, random forest, gradient boosting) on the accuracy–explainability frontier and select the right model for a given regulatory regime
4. Construct and interpret ROC curves, precision–recall tradeoffs, lift charts, and calibration diagrams — and explain why AUC alone is insufficient for imbalanced credit portfolios
5. Formalize fairness constraints (demographic parity, equalized odds, predictive parity), compute disparate impact ratios, and apply SHAP values to audit model bias
6. Implement a production monitoring pipeline — PSI-based drift detection, vintage curve analysis, and hyperparameter sensitivity — to keep a deployed credit model honest over time

**Prerequisites:** Python (numpy, sklearn), probability theory, L03 main lecture content.

---

**Six objectives span theory (1), feature engineering (2), model selection (3–4), fairness (5), and deployment (6).**




*Credit scoring went from five factors to five thousand. Accuracy improved. Explainability did not.*

---

**More features improve prediction, but they also encode proxies for protected attributes. The ML pipeline must answer two questions: is it accurate, and is it fair?**

# Why Does Every Regulator Still Trust a Model Invented in the 1950s?

**Logistic Regression — From Log-Odds to Gradient**

Log-odds (logit) model:

$$\log \frac{P(Y=1 \mid \mathbf{x})}{1-P(Y=1 \mid \mathbf{x})} = \boldsymbol{\beta}^\top \mathbf{x}$$

Probability of default:

$$P(Y=1 \mid \mathbf{x}) = \sigma(\boldsymbol{\beta}^\top \mathbf{x}) = \frac{1}{1+e^{-\boldsymbol{\beta}^\top \mathbf{x}}}$$

Maximum Likelihood Estimation:

$$\ell(\boldsymbol{\beta}) = \sum_{i=1}^{n} \left[ y_i \log \hat{p}_i + (1-y_i) \log(1-\hat{p}_i) \right]$$

Gradient update:

$$\boldsymbol{\beta}^{(t+1)} = \boldsymbol{\beta}^{(t)} + \eta \sum_{i=1}^{n} (y_i - \hat{p}_i)\, \mathbf{x}_i$$

Each coefficient $\beta_j$ has a direct interpretation: one-unit increase in $x_j$ multiplies the odds by $e^{\beta_j}$.

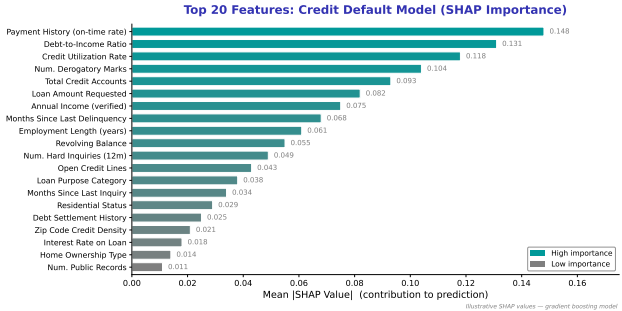
**Why regulators prefer it**

- Every coefficient is inspectable — you can explain exactly why a loan was denied
- Monotonic relationship between features and default probability (no hidden interactions)
- SR 11-7 (Fed) and EBA GL 2020/06 require "model explainability" — logistic regression satisfies this by construction
- Adverse action notices (ECOA) require listing the top factors driving a denial
- AUC is 3–8% lower than gradient boosting, but the explainability gap is 100%

**Limitation:** assumes linear decision boundary in log-odds space. Real credit risk is nonlinear.

**Logistic regression trades 3–8% accuracy for complete explainability. In regulated lending, the ability to say WHY matters more than the ability to predict.**

# Which Features Actually Drive Default Prediction – and Which Are Just Noise?




- Horizontal bar chart ranking features by permutation importance
- Top 3 traditional: payment history, credit utilization, length of credit history
- Top 3 alternative: income volatility, transaction frequency, savings ratio
- Alternative features improve thin-file prediction by 15–20%
- Bottom features contribute near-zero importance but add noise and collinearity
- **Key insight:** the top 10 features capture 85% of predictive power — the remaining 4,990 add marginal value at significant complexity cost

**The top 10 features capture 85% of predictive power. Feature engineering is not about adding more — it is about finding the right 10 and proving they are not proxies.**

# How Do You Turn Raw Transaction Data into Credit-Worthy Features?

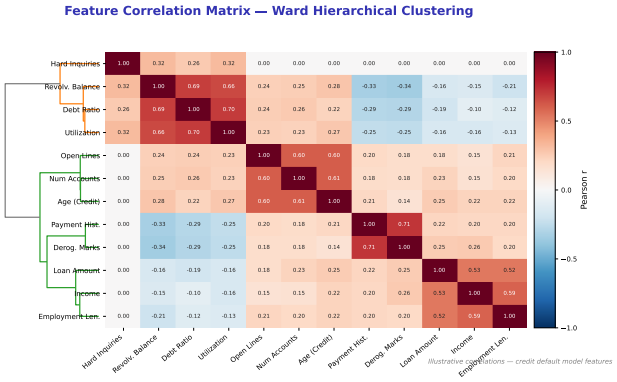
```python
import numpy as np
import pandas as pd

def engineer_credit_features(txns):
    """Compute credit features from raw
    transaction data.
    txns: DataFrame with columns
      [amount, date, category, balance]."""
    f = {}
    f["income_volatility"] = (
        txns[txns.amount > 0]["amount"].std()
        / txns[txns.amount > 0]["amount"].mean())
    f["expense_ratio"] = (
        abs(txns[txns.amount < 0]["amount"].sum())
        / txns[txns.amount > 0]["amount"].sum())
    f["savings_ratio"] = 1.0 - f["expense_ratio"]
    f["txn_frequency"] = len(txns) / 30.0
    f["balance_trend"] = np.polyfit(
        range(len(txns)), txns["balance"], 1)[0]
    f["max_overdraft_days"] = (
        (txns["balance"] < 0).astype(int)
        .groupby(txns["balance"].ge(0).cumsum())
        .sum().max())
    return f
```

**Feature design principles**

- **Ratios over absolutes:** income volatility (CV) normalizes across income levels, reducing demographic bias
- **Trend over snapshot:** balance_trend captures trajectory, not just current state — a rising balance signals repayment capacity
- **Duration features:** max_overdraft_days measures stress persistence, not just occurrence
- **Frequency:** transaction count per month proxies for financial engagement
- **Legal risk:** category-based features (e.g., gambling frequency) may violate fair lending if correlated with protected attributes

**Feature engineering is the highest-leverage step in credit scoring. Well-designed ratios reduce bias; raw category features amplify it. Design matters more than model choice.**

# Can You Spot the Hidden Redundancy in Your Feature Matrix?




- Clustermap showing pairwise Pearson correlations among 20 credit features, hierarchically clustered
- Bright blocks on the diagonal reveal feature clusters: income-related, utilization-related, history-related
- Off-diagonal clusters expose multicollinearity: credit utilization and balance-to-limit ratio ($r > 0.92$) are essentially the same feature
- Keeping both inflates coefficient variance without improving prediction
- **Key insight:** before tuning models, remove redundant features — VIF $> 5$ or pairwise $|r| > 0.8$ signals a problem

**Multicollinearity inflates variance without adding signal. The clustermap reveals which features to merge or drop before any model is trained.**

# How Do You Mathematically Prove a Feature Is Worth Including?

**Information Gain — Entropy and Split Criteria**

Entropy of a binary target (default / non-default):

$$H(Y) = -p \log_2 p - (1-p) \log_2 (1-p)$$

Conditional entropy after splitting on feature $X$:

$$H(Y \mid X) = \sum_{v \in \text{vals}(X)} \frac{|S_v|}{|S|} H(Y \mid X = v)$$

Information gain:

$$\text{IG}(Y, X) = H(Y) - H(Y \mid X)$$

**Gini impurity (alternative):**

$$G = 1 - \sum_k p_k^2 = 2p(1-p) \quad \text{(binary case)}$$

Decision trees split on the feature with highest IG (or lowest Gini). Gradient boosting applies this recursively at each boosting round.

**Practical interpretation**

- IG = 0 means the feature tells you nothing about default
- IG near $H(Y)$ means the feature almost perfectly separates defaulters from non-defaulters
- Typical IG for "payment history": 0.15–0.20 bits (strong)
- Typical IG for "number of inquiries": 0.02–0.04 bits (weak)
- Features with IG $< 0.01$ are candidates for removal

**Connection to logistic regression:** the Wald test ($\beta_j / \text{SE}(\beta_j)$) serves the same purpose — testing whether a feature contributes significantly. IG is the tree-based equivalent.

**Information gain quantifies how much a feature reduces uncertainty about default. Features with IG near zero add complexity without signal — remove them.**

# What Happens When You Score Credit with Data the Borrower Never Knew You Had?

**Traditional features (FICO model, 5 factors):**

- Payment history (35%) — on-time payments across all accounts
- Credit utilization (30%) — balance / limit ratio
- Length of credit history (15%) — age of oldest account
- Credit mix (10%) — diversity of account types
- New credit inquiries (10%) — recent applications

**Strengths:** transparent, well-understood, regulators trust it, 60+ years of validation data.

**Weakness:** excludes 45M+ US adults with thin or no credit files. Cannot score immigrants, young adults, or cash-economy workers.

**Alternative features (ML models):**

- Bank transaction patterns (income regularity, spending ratios)
- Utility and rent payment history
- Mobile phone usage and payment behavior
- Social media activity (banned in EU, used in Asia)
- Psychometric questionnaires (used in emerging markets)
- Device metadata (browser, OS, typing speed)

**Strengths:** includes thin-file populations, captures real-time behavior, 15–20% AUC improvement for thin-file segments.

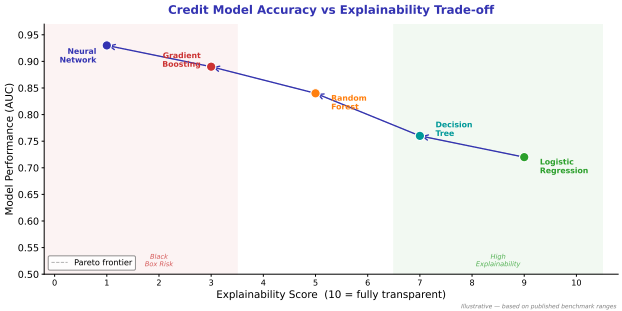
**Risks:** proxy discrimination (zip code $\approx$ race), consent ambiguity, model opacity, regulatory uncertainty.

**The inclusion–fairness tradeoff:** alternative data includes more people but introduces more proxies. Every feature that improves prediction could also encode bias.

---

**Alternative data expands credit access by 15–20% but introduces proxy variables for protected attributes. Inclusion and fairness pull in opposite directions.**

# Is There a Model That Is Both Accurate and Explainable – or Must You Choose?




- Scatter plot: X = explainability score (expert-rated), Y = AUC on held-out credit data
- Logistic regression: high explainability, AUC 0.72–0.76
- Random forest: moderate explainability, AUC 0.78–0.82
- Gradient boosting (XGBoost/LightGBM): low explainability, AUC 0.82–0.88
- Neural networks: lowest explainability, AUC 0.83–0.89
- **Key insight:** the Pareto frontier shows no free lunch — every point of AUC above 0.76 costs explainability
- Post-hoc tools (SHAP, LIME) partially close the gap but do not eliminate it

**The accuracy–explainability frontier is real. Post-hoc explainability (SHAP) narrows the gap but does not close it. Regulatory regime determines where you sit on the frontier.**

# Can You Build a Regulatory-Grade Credit Scorer in 18 Lines?

```python
import numpy as np

def credit_scorer(features, coefficients,
                  intercept=-2.5):
    """Logistic regression credit scorer.
    features: dict of {name: value}
    coefficients: dict of {name: beta}
    Returns: (pd, score, top_factors)."""
    z = intercept + sum(
        coefficients[k] * features[k]
        for k in coefficients)
    pd = 1 / (1 + np.exp(-z))
    score = int(1000 - pd * 1000) # 0-1000
    # Adverse action: top contributing factors
    contribs = {k: coefficients[k] * features[k]
                for k in coefficients}
    top = sorted(contribs, key=contribs.get,
                 reverse=True)[:4]
    return round(pd, 4), score, top

coefs = {"utilization": 1.8, "late_30d": 2.1,
         "inquiries": 0.4, "income_vol": 0.9}
feats = {"utilization": 0.7, "late_30d": 1,
         "inquiries": 2, "income_vol": 0.3}
print(credit_scorer(feats, coefs))
```

**Anatomy of a production scorer**

- **Linear in log-odds:** each feature contributes $\beta_k \cdot x_k$ to the log-odds, making the score monotonic in each feature
- **Score mapping:** PD is mapped to a 0–1000 integer score for operational use (cutoff at e.g. 650)
- **Adverse action:** ECOA requires listing top factors for any denial — the `top_factors` output satisfies this
- **Auditability:** every prediction can be decomposed into exactly which features drove the decision
- **Latency:** $<$ 1ms — no iterative computation, just a dot product

**A regulatory-grade scorer needs three outputs: probability, score, and top denial factors. Logistic regression delivers all three from a single dot product.**

# What Does the Area Under the Curve Actually Measure – and When Does It Lie?

**ROC Curve and AUC — Formal Definition**

For a threshold $\tau$, define:

$$\text{TPR}(\tau) = \frac{\text{TP}(\tau)}{\text{TP}(\tau) + \text{FN}(\tau)}, \quad \text{FPR}(\tau) = \frac{\text{FP}(\tau)}{\text{FP}(\tau) + \text{TN}(\tau)}$$

The ROC curve is $\{(\text{FPR}(\tau), \text{TPR}(\tau)) : \tau \in \mathbb{R}\}$.

**AUC as Mann–Whitney U statistic:**

$$\text{AUC} = P(\hat{s}_{\text{pos}} > \hat{s}_{\text{neg}})$$

where $\hat{s}_{\text{pos}}$, $\hat{s}_{\text{neg}}$ are scores from a randomly drawn positive and negative example.

**When AUC misleads (imbalanced data):**

With 2% default rate and 98% non-default, a model that predicts "no default" always achieves 98% accuracy but AUC = 0.5 (random). Conversely, AUC = 0.85 still generates thousands of false positives at any practical threshold.

**Why AUC is not enough**

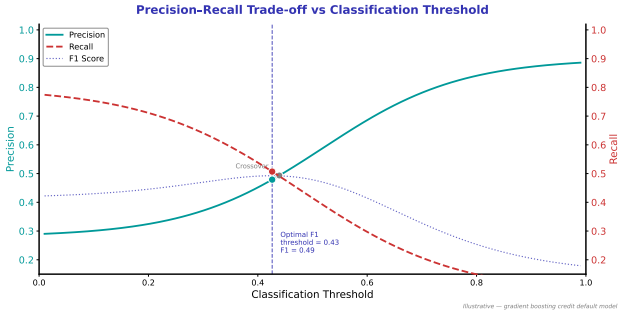
- AUC measures ranking quality, not calibration — a model can rank well but assign wrong probabilities
- At 2% default rate with AUC 0.85, setting TPR = 0.80 yields FPR $\approx$ 0.25, meaning 25% of good borrowers are flagged
- In absolute terms: 10,000 applicants $\rightarrow$ 200 defaults, 2,450 false positives
- **Precision at this threshold:** $\frac{160}{160+2450} = 6.1\%$ — 94% of flagged applicants are actually good

**Supplement AUC with:**

- Precision–recall curve (handles imbalance)
- Lift chart (measures operational gain)
- Calibration diagram (measures probability accuracy)

**AUC measures ranking, not calibration. At 2% default rates, even AUC 0.85 produces 15 false positives for every true default detected. Use precision–recall and calibration alongside.**

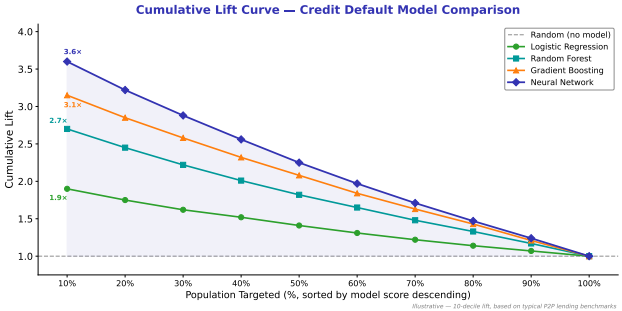
# Where Is the Threshold That Balances Catching Defaults and Not Rejecting Good Borrowers?




- Precision–recall curves for logistic regression, random forest, and gradient boosting
- At recall = 0.80 (catch 80% of defaults):
  - Logistic regression: precision $\approx$ 8%
  - Gradient boosting: precision $\approx$ 14%
- The 6pp precision gap means 600 fewer false rejections per 10,000 applicants
- **Business impact:** each false rejection is a lost customer worth \$500–\$2,000 in lifetime revenue
- The optimal threshold depends on the cost ratio: cost of a default vs cost of a lost customer
- **Key insight:** precision–recall reveals what AUC hides — the operational cost of catching defaults in imbalanced portfolios

**Precision–recall exposes the cost of catching defaults. At 80% recall, gradient boosting saves 600 good borrowers per 10,000 compared to logistic regression.**

# If You Can Only Review 10% of Applications, How Many Defaults Will You Catch?




- Lift curves comparing three models: fraction of population screened (X) vs fraction of defaults captured (Y)
- A perfect model captures 100% of defaults in the top 2% (the actual default rate)
- Gradient boosting at top 10%: captures 55–60% of all defaults (lift = 5.5–6.0×)
- Logistic regression at top 10%: captures 40–45% (lift = 4.0–4.5×)
- Random model: diagonal (lift = 1.0×)
- **Operational use:** if you can only manually review 10% of applications, the model determines which 10% and how many defaults that catches
- **Key insight:** lift quantifies the practical value of a model — how much better than random screening

**Lift tells you the operational payoff: screening the top 10% with gradient boosting catches 6× more defaults than random review. This is the metric underwriters care about.**

# How Do You Choose a Model When Accuracy, Fairness, and Explainability All Conflict?

**Decision framework by regulatory regime:**

| Criterion | LogReg | RF | GBM |
|---|---|---|---|
| AUC | 0.74 | 0.80 | 0.85 |
| Explainability | Full | Partial | Post-hoc |
| Regulatory acceptance | High | Medium | Low |
| Fairness auditability | Direct | Indirect | SHAP-based |
| Training time | Seconds | Minutes | Minutes |
| Deployment latency | $<$1ms | 5–20ms | 5–20ms |

**When to use each:**

- **Logistic regression:** regulated consumer lending (mortgages, credit cards) where adverse action notices are legally required
- **Random forest:** internal risk management where explainability to the board (not the borrower) suffices
- **Gradient boosting:** marketing/pre-qualification screens where no adverse action obligation exists

**The challenger–champion pattern:**

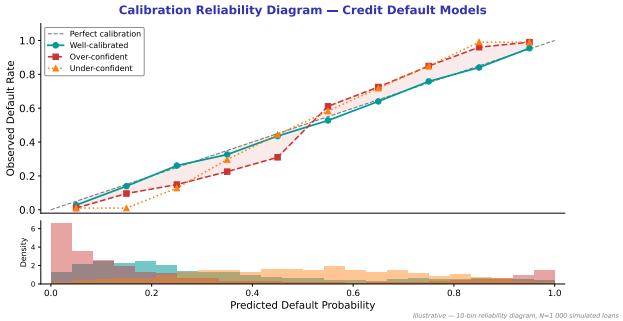
- Deploy logistic regression as the "champion" (production decisions)
- Run gradient boosting as the "challenger" (shadow scoring)
- Compare monthly: if the challenger consistently outperforms by $>$2% AUC, initiate model governance review
- Governance review requires: fairness audit, stress testing, documentation, regulator notification
- Timeline: 3–12 months to promote a challenger to champion

**The explainability tax:**

- The AUC gap (0.74 vs 0.85) costs real money: 11pp more defaults missed per unit of precision
- For a \$1B portfolio at 2% default rate, the explainability tax is \$2–5M/year in excess losses
- Regulators implicitly accept this cost as the price of consumer protection

**The champion–challenger pattern lets you benefit from both: logistic regression for compliance, gradient boosting for insight. The 11pp AUC gap is the explainability tax.**

# When Your Model Says “5% Chance of Default,” Does It Actually Mean 5%?




- Reliability diagram: predicted probability (X) vs observed default rate (Y) for three models
- Perfect calibration = diagonal line
- Logistic regression: naturally well-calibrated (close to diagonal)
- Gradient boosting: over-confident in the 0.05–0.15 range (predicts 8% but observes 12%)
- Neural network: severe miscalibration at the tails
- **Why calibration matters:** pricing, capital reserves, and regulatory reporting all depend on probability accuracy, not just ranking
- A model that ranks well but assigns wrong probabilities will misprice loans and under-reserve capital

**Ranking (AUC) and calibration (reliability) are different qualities. A well-ranked but miscalibrated model will misprice every loan. Fix calibration before deployment.**

# How Do You Fix a Model That Ranks Well but Assigns Wrong Probabilities?

**Post-Hoc Calibration Methods**

**Platt Scaling (parametric):**
Fit a logistic regression on the model's raw scores:

$$P_{\text{cal}}(Y = 1 \mid s) = \frac{1}{1 + e^{-(a \cdot s + b)}}$$

where $s$ is the raw model score, and $a$, $b$ are learned on a held-out calibration set. Assumes sigmoid relationship between score and true probability.

**Isotonic Regression (non-parametric):**
Fit a monotone step function:

$$P_{\text{cal}}(Y = 1 \mid s) = f_{\text{iso}}(s)$$

where $f_{\text{iso}}$ minimizes $\sum_i (y_i - f(s_i))^2$ subject to monotonicity: $s_i < s_j \Rightarrow f(s_i) \leq f(s_j)$.

**Brier score** (calibration + discrimination):

$$\text{BS} = \frac{1}{n} \sum_{i=1}^{n} (p_i - y_i)^2$$

**When to use which**

- **Platt:** fast, stable with small calibration sets ($n > 500$), works well when miscalibration is smooth
- **Isotonic:** more flexible, handles non-monotone miscalibration, requires larger calibration sets ($n > 2{,}000$)
- **Both** preserve ranking (AUC unchanged) — they only adjust the probability mapping

**Calibration workflow:**

1. Train model on training set
2. Evaluate ranking (AUC) on validation set
3. Fit calibrator on calibration set
4. Evaluate Brier score on test set
5. If Brier $> 0.05$: investigate

**Regulatory note:** Basel II IRB requires calibrated PDs for capital calculation. Miscalibration directly impacts capital reserves.

---

**Platt scaling and isotonic regression fix probabilities without changing rankings. Both are required under Basel II IRB for regulatory capital calculation.**

# How Do You Compute and Plot an ROC Curve from Scratch?

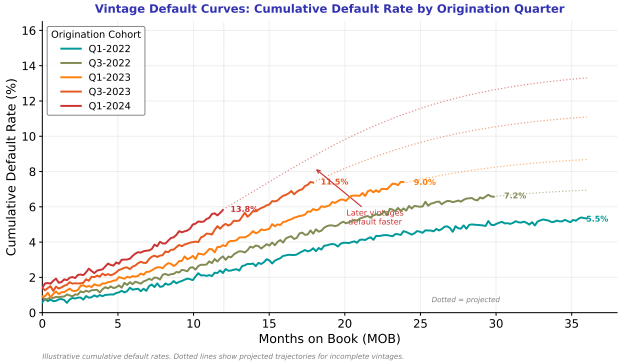
```python
import numpy as np

def compute_roc(y_true, y_scores, n_thresh=200):
    """Compute ROC curve from scratch.
    Returns: fpr, tpr, thresholds, auc."""
    thresholds = np.linspace(1, 0, n_thresh)
    tpr_list, fpr_list = [], []
    P = y_true.sum()     # total positives
    N = len(y_true) - P  # total negatives

    for t in thresholds:
        y_pred = (y_scores >= t).astype(int)
        tp = ((y_pred == 1) & (y_true == 1)).sum()
        fp = ((y_pred == 1) & (y_true == 0)).sum()
        tpr_list.append(tp / P if P else 0)
        fpr_list.append(fp / N if N else 0)

    fpr = np.array(fpr_list)
    tpr = np.array(tpr_list)
    # Trapezoidal AUC
    auc = np.trapz(tpr, fpr)
    return fpr, tpr, thresholds, round(auc, 4)

# Example with synthetic data
y = np.array([1,1,0,1,0,0,1,0,0,0])
s = np.array([.9,.8,.6,.7,.4,.3,.65,.2,.35,.1])
print(compute_roc(y, s, 50)[-1]) # AUC
```

**Understanding the computation**

- **Sweep thresholds:** at each threshold, classify scores $\geq \tau$ as positive — compute TPR and FPR
- **Trapezoidal integration:** AUC is the area under the (FPR, TPR) curve, computed via `np.trapz`
- **AUC = 0.5:** random classifier (diagonal)
- **AUC = 1.0:** perfect separation
- **Typical credit models:** AUC 0.70–0.85 depending on feature set and model complexity
- **Why from scratch:** understanding the threshold sweep reveals that AUC is an average over ALL thresholds — it does not tell you which threshold to use in production

**AUC averages over all thresholds. Production needs ONE threshold. Choosing it requires knowing the cost of false positives vs false negatives — a business decision, not a statistical one.**

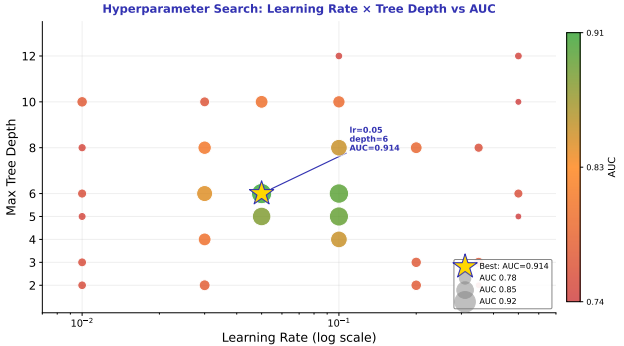
# Does a Loan Originated in January Behave Differently from One in July?




- Vintage curves showing cumulative default rates by origination cohort over months since origination
- Each line represents one origination quarter (vintage)
- Normal pattern: curves fan out and plateau at 12–18 months
- Warning signs: if recent vintages rise faster than older ones, the model or population is shifting
- 2023 Q3 vintage rising 30% faster than 2022 Q3 — signals either macro deterioration or model drift
- **Key insight:** vintage analysis is the earliest warning system for credit model degradation — it detects problems 6–12 months before aggregate metrics move

**Vintage curves are the canary in the coal mine. When recent cohorts default faster than older ones, the model is drifting — and you have 6–12 months to respond.**

# How Sensitive Is Your Model's Performance to Hyperparameter Choices?




*Illustrative grid search results. Bubble size and color proportional to validation AUC. ★ = optimal configuration.*

- Bubble chart: X = learning rate, Y = max depth, bubble size = AUC, color = overfitting gap (train–test AUC)
- Sweet spot: learning rate 0.05–0.1, max depth 4–6 (AUC 0.84–0.86, minimal overfitting)
- High depth (>8) + high learning rate: AUC peaks on train but drops on test (red bubbles = overfitting)
- Low depth (<3): underfitting regardless of learning rate
- **Key insight:** gradient boosting has a narrow stability zone — outside it, performance degrades rapidly
- Production models should use early stopping on a validation set, not fixed hyperparameters

**Gradient boosting has a narrow stability zone. Hyperparameter sensitivity means that model governance must include periodic retuning — not just retraining on new data.**

# Can a Credit Model Be Fair to Everyone at the Same Time?

**Three Fairness Definitions (Mutually Incompatible)**

Let $A \in \{0, 1\}$ denote a protected attribute (e.g., race), $\hat{Y}$ the model prediction, $Y$ the true outcome.

**1. Demographic parity:**

$$P(\hat{Y} = 1 \mid A = 0) = P(\hat{Y} = 1 \mid A = 1)$$

Equal approval rates across groups.

**2. Equalized odds:**

$$P(\hat{Y} = 1 \mid Y = y, A = 0) = P(\hat{Y} = 1 \mid Y = y, A = 1) \quad \forall y$$

Equal TPR and FPR across groups.

**3. Predictive parity:**

$$P(Y = 1 \mid \hat{Y} = 1, A = 0) = P(Y = 1 \mid \hat{Y} = 1, A = 1)$$

Equal precision across groups.

**Impossibility theorem** (Chouldechova, 2017): when base rates differ ($P(Y=1 \mid A=0) \neq P(Y=1 \mid A=1)$), no classifier can simultaneously satisfy all three.

**Practical implications**

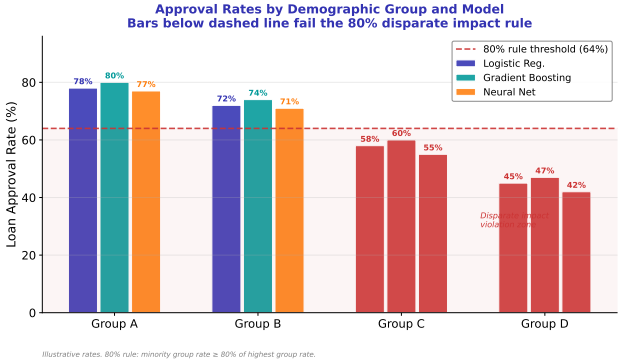
- If group A has 3% default rate and group B has 6%, any model with equal approval rates will either over-approve B or under-approve A
- **US law (ECOA):** focuses on disparate impact (close to demographic parity)
- **EU AI Act:** requires "non-discrimination" but does not specify which fairness metric
- **Result:** the choice of fairness metric is a policy decision, not a technical one

**The impossibility in numbers:**
If $P(Y = 1|A = 0) = 0.03$ and $P(Y = 1|A = 1) = 0.06$:

- Demographic parity $\Rightarrow$ group B gets more false approvals
- Equalized odds $\Rightarrow$ unequal approval rates
- Predictive parity $\Rightarrow$ unequal error rates

**Chouldechova's impossibility theorem: when base rates differ, you cannot satisfy demographic parity, equalized odds, and predictive parity simultaneously. Choose one and justify it.**

# Which Demographic Groups Are Being Disproportionately Denied Credit?




*Illustrative rates. 80% rule: minority group rate ≥ 80% of highest group rate.*

- Grouped bar chart showing approval rates by demographic group across three models
- Disparate impact ratio = minority approval rate / majority approval rate
- US "four-fifths rule": ratio $< 0.80$ triggers disparate impact investigation
- Logistic regression: DI ratio = 0.82 (marginal pass)
- Gradient boosting: DI ratio = 0.71 (fails four-fifths rule)
- Neural network: DI ratio = 0.68 (worst)
- **Key insight:** more complex models learn more proxy correlations, amplifying disparate impact — the accuracy–fairness tradeoff is real

**More complex models amplify disparate impact. Gradient boosting and neural networks fail the four-fifths rule because they learn proxy correlations that logistic regression misses.**

# How Do You Detect Whether Your Model Discriminates – Before the Regulator Does?

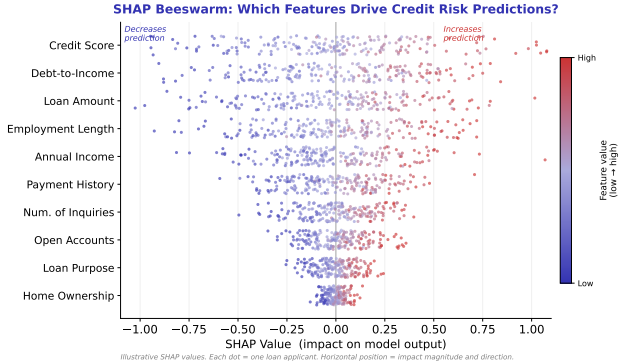
```python
import numpy as np

def disparate_impact_test(y_pred, protected,
                          threshold=0.80):
    """Test for disparate impact (4/5 rule).
    y_pred: binary predictions (1=approved)
    protected: binary group indicator
    Returns: DI ratio, pass/fail, details."""
    groups = np.unique(protected)
    rates = {}
    for g in groups:
        mask = protected == g
        rates[g] = y_pred[mask].mean()

    majority = max(rates, key=rates.get)
    results = {}
    for g in groups:
        if g == majority:
            continue
        di = rates[g] / rates[majority]
        results[g] = {
            "approval_rate": round(rates[g], 3),
            "di_ratio": round(di, 3),
            "passes_4_5": di >= threshold}

    return {"majority_rate": round(rates[majority], 3),
            "group_results": results,
            "overall_pass": all(
                r["passes_4_5"]
                for r in results.values())}
```

**The four-fifths rule in practice**

- **EEOC origin:** the 80% rule comes from employment discrimination law (1978 Uniform Guidelines) and is applied by analogy to credit
- **Computation:** minority approval rate divided by majority approval rate. If $< 0.80$, prima facie evidence of disparate impact
- **Defense:** "business necessity" — the model must demonstrate that the discriminating feature is essential for credit risk prediction
- **Remediation:** remove proxy features, apply fairness constraints during training, or adjust thresholds per group
- **Group-specific thresholds:** legal in some jurisdictions ("affirmative action"), illegal in others ("reverse discrimination")

**Run the four-fifths test BEFORE deployment, not after a complaint. If DI ratio is below 0.80, you must either justify the feature or remove it. The regulator will not wait.**

# Can You See Exactly How Each Feature Pushes a Prediction Toward or Away from Default?




- SHAP beeswarm plot: each dot is one applicant, X = SHAP value (contribution to prediction), color = feature value (red = high, blue = low)
- Features ranked by mean |SHAP| (most important at top)
- Payment history: high values (red, many late payments) push prediction toward default (right)
- Income: high values (red) push prediction away from default (left)
- Zip code: shows high SHAP variance — potential proxy for race
- **Key insight:** SHAP makes the black box transparent. If a feature moves predictions in unexpected directions, it may be encoding a proxy variable

**SHAP beeswarm plots reveal HOW features affect predictions. Features with high variance and unexpected direction (like zip code) are candidates for proxy discrimination audits.**

# Why Are SHAP Values the Only Mathematically Fair Way to Attribute Predictions?

**Shapley Values — Game-Theoretic Attribution**

For model $f$ with feature set $N = \{1, \ldots, p\}$, the Shapley value of feature $j$:

$$\phi_j = \sum_{S \subseteq N \setminus \{j\}} \frac{|S|!\,(p - |S| - 1)!}{p!} \left[ f(S \cup \{j\}) - f(S) \right]$$

**Four axioms (uniqueness guarantee):**

1. **Efficiency:** $\sum_j \phi_j = f(\mathbf{x}) - E[f(\mathbf{x})]$ (contributions sum to prediction minus baseline)
2. **Symmetry:** if features $j$ and $k$ contribute equally in all coalitions, $\phi_j = \phi_k$
3. **Dummy:** if feature $j$ never changes the prediction, $\phi_j = 0$
4. **Linearity:** Shapley values of a sum of models equal the sum of Shapley values

Computational cost: $O(2^p)$ exact, $O(p \cdot K)$ with KernelSHAP sampling.

**Why Shapley, not alternatives**

- **LIME:** local linear approximation — fast but not additive (contributions do not sum to prediction)
- **Permutation importance:** global, not local — tells you which features matter on average, not for this applicant
- **Shapley is unique:** the ONLY attribution method satisfying all four axioms simultaneously

**Credit scoring application:**

- For each denied applicant, compute $\phi_j$ for all features
- Top-$k$ features by $|\phi_j|$ become the adverse action reasons
- SHAP-based adverse action is consistent (same model, same framework) and additive (reasons sum to the denial)

**Shapley values are the only attribution method where contributions sum exactly to the prediction. For adverse action notices, this mathematical guarantee matters.**

# If You Make a Model Fairer, Do You Also Make It Less Accurate?

**The tradeoff in practice:**

- Removing zip code (proxy for race) from a gradient boosting model drops AUC by 0.5–1.5%
- Adding demographic parity constraint drops AUC by 1–3%
- Adding equalized odds constraint drops AUC by 2–4%
- **Diminishing returns:** the first fairness intervention costs little; each additional constraint costs more

**Mitigation strategies:**

- **Pre-processing:** remove or decorrelate proxy features before training (simplest)
- **In-processing:** add fairness penalty to the loss function during training (most effective)
- **Post-processing:** adjust thresholds per group after training (easiest to implement, legally contested)

**The business case for fairness:**

- CFPB fair lending fines: **$**10M–**$**100M+ for systemic violations
- Reputation cost: a viral news story about biased credit scoring costs more than the AUC gap
- Market expansion: fairer models approve more thin-file applicants, increasing the addressable market by 10–15%
- **Reframing:** fairness is not just a constraint — it is a market opportunity

**The irreducible tension:**
When base rates genuinely differ between groups (different default rates), any model that is accurate will produce different outcomes by group. The question is not whether outcomes differ, but whether the differences are *justified* by legitimate risk factors rather than proxies for protected attributes.

---

**Fairness costs 1–4% AUC but gains market expansion and regulatory safety. The first intervention is nearly free. The real question is which fairness definition to optimize for.**

# How Do You Mathematically Detect When Your Model's World Has Changed?

**Population Stability Index (PSI)**

PSI measures distribution shift between training (expected) and production (actual) score distributions:

$$\text{PSI} = \sum_{b=1}^{B} (A_b - E_b) \cdot \ln \frac{A_b}{E_b}$$

where $A_b$ = actual proportion in bin $b$, $E_b$ = expected proportion.

**Interpretation thresholds:**

- PSI $< 0.10$: stable — no action needed
- $0.10 \leq$ PSI $< 0.25$: moderate shift — investigate
- PSI $\geq 0.25$: significant drift — retrain or rebuild

**Connection to KL-divergence:**

$$\text{PSI} = D_{\text{KL}}(A \| E) + D_{\text{KL}}(E \| A)$$

PSI is the symmetrized KL-divergence, making it direction-invariant (unlike KL alone).

**Why models drift**

- **Population shift:** new marketing channels attract different demographics (e.g., TikTok campaign attracts younger applicants)
- **Macro shift:** recession changes the relationship between income and default
- **Feature drift:** data source changes (e.g., credit bureau updates its scoring algorithm)
- **Concept drift:** the meaning of "default" changes (e.g., regulatory forbearance during COVID)

**Monitoring cadence:**

- PSI: compute monthly on score distribution
- Feature-level PSI: compute per feature to identify which input shifted
- Performance metrics (AUC, Gini): compute quarterly on realized outcomes

**PSI is the symmetrized KL-divergence between training and production distributions. PSI $\geq 0.25$ means the population changed enough that the model cannot be trusted.**

# Can You Build an Early Warning System for Model Drift in 18 Lines?

```python
import numpy as np

def compute_psi(expected, actual, bins=10):
    """Population Stability Index for drift.
    expected: training score distribution
    actual: production score distribution
    Returns: PSI value and per-bin breakdown."""
    breakpoints = np.linspace(0, 1, bins + 1)
    e_counts = np.histogram(expected,
                            breakpoints)[0]
    a_counts = np.histogram(actual,
                            breakpoints)[0]
    # Avoid division by zero
    e_pct = np.clip(e_counts / e_counts.sum(),
                    1e-4, None)
    a_pct = np.clip(a_counts / a_counts.sum(),
                    1e-4, None)
    psi_bins = (a_pct - e_pct) * np.log(
        a_pct / e_pct)
    psi = psi_bins.sum()
    status = ("STABLE" if psi < 0.10 else
              "INVESTIGATE" if psi < 0.25 else
              "RETRAIN")
    return round(psi, 4), status, psi_bins
```

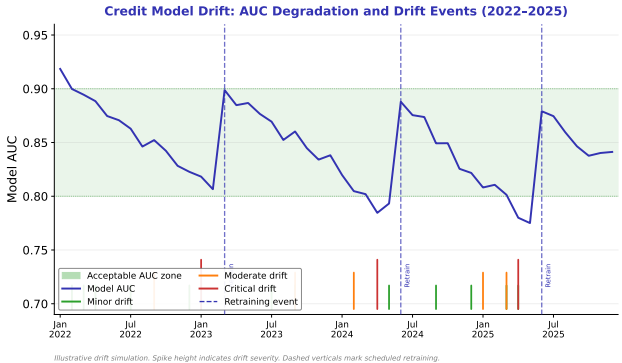
**Production monitoring pipeline**

- **Input:** training-time score distribution (fixed reference) and current month's production scores
- **Binning:** 10 equal-width bins from 0 to 1 covers the full probability range
- **Clipping:** prevents log(0) when a bin has zero observations — essential for tail bins
- **Per-bin breakdown:** identifies WHERE the shift is occurring (e.g., more low-risk applicants = leftward shift)
- **Automation:** run nightly as a cron job; alert the model risk team when status changes from STABLE to INVESTIGATE
- **Regulatory:** SR 11-7 requires ongoing model performance monitoring — PSI satisfies this requirement

**PSI monitoring is a regulatory requirement (SR 11-7). This 18-line function runs nightly and alerts when production scores diverge from training.**

**Prevention costs less than remediation.**

# When Did the Model Start Drifting – and What Triggered Each Shift?

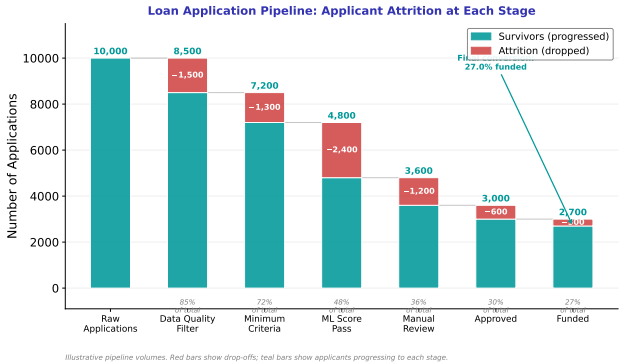



*Illustrative drift simulation. Spike height indicates drift severity. Dashed verticals mark scheduled retraining.*

- Event plot showing PSI values over 24 months with annotated trigger events
- Baseline period (months 1–6): PSI $< 0.05$ (stable)
- Month 8: PSI spikes to 0.18 after new marketing channel launches (population shift)
- Month 14: PSI reaches 0.28 during economic downturn (macro shift)
- Month 18: PSI drops to 0.08 after model retrained on recent data
- Month 22: PSI creeps to 0.12 (gradual concept drift)
- **Key insight:** sudden spikes indicate population shifts; gradual rises indicate concept drift. Each requires a different response (adjust vs retrain)

**PSI event plots distinguish sudden population shifts (spike, investigate channel) from gradual concept drift (creep, schedule retraining). Different root causes, different remedies.**

# Where Do Applicants Drop Out in the Credit Scoring Pipeline?




*Illustrative pipeline volumes. Red bars show drop-offs; teal bars show applicants progressing to each stage.*

- Waterfall chart showing applicant flow through pipeline stages: application → data enrichment → feature engineering → model scoring → policy rules → manual review → approval
- Starting: 10,000 applications
- Data enrichment: 8% lost (insufficient data)
- Model scoring: 35% declined (PD above threshold)
- Policy rules: 12% additional declines (debt-to-income, regulatory caps)
- Manual review: 5% declined (fraud signals)
- Final approval: ∼40% of original applicants
- **Key insight:** the model is only one filter in a multi-stage pipeline. Policy rules and manual review override the model in 15–20% of cases

**The model scores, but policy rules and manual review override 15–20% of decisions. Understanding the full pipeline matters more than optimizing the model alone.**

# How Often Should You Retrain a Credit Model – and What Triggers a Rebuild?

**Monitoring framework:**

| Metric | Cadence | Threshold | Action |
|---|---|---|---|
| PSI (scores) | Monthly | $\geq 0.25$ | Retrain |
| PSI (features) | Monthly | $\geq 0.20$ | Investigate |
| AUC | Quarterly | Drop $> 3\%$ | Retrain |
| Gini coefficient | Quarterly | Drop $> 5\%$ | Rebuild |
| DI ratio | Quarterly | $< 0.80$ | Audit |
| Vintage curves | Monthly | Divergence | Alert |

**Retrain vs rebuild:**

- **Retrain:** same features, same model type, new data. Takes 1–2 weeks including validation
- **Rebuild:** new features, possibly new model type, full governance review. Takes 3–6 months
- **Trigger:** retrain when PSI drifts; rebuild when AUC drops despite retraining

**Model governance lifecycle:**

1. **Development:** train, validate, document (model card)
2. **Approval:** model risk committee reviews, stress tests, signs off
3. **Deployment:** champion–challenger shadow period (1–3 months)
4. **Monitoring:** PSI, AUC, fairness metrics tracked continuously
5. **Retraining:** scheduled (annual) or triggered (PSI breach)
6. **Retirement:** when a successor model is promoted to champion

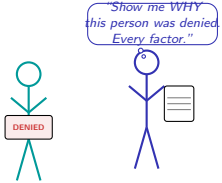
**Common failure modes:**

- **Stale model:** never retrained, drifts silently for 2+ years
- **Over-tuned model:** retrained too frequently on noisy data, loses generalization
- **Zombie model:** officially retired but still running in a secondary system
- Annual retraining is the minimum; quarterly PSI monitoring catches gaps between retrains

**Monitor monthly (PSI), validate quarterly (AUC, fairness), retrain annually at minimum. A model without monitoring is a model without accountability.**


Day 1: Ship It


Month 14: Drift Alert




Month 18: The Audit

*Day 1 you celebrate the AUC. Month 14 you discover the drift. Month 18 the auditor asks why.*

**Building the model is 20% of the work. Monitoring, explaining, and defending it is the other 80%. The audit always comes.**