Week 8: Fraud Detection as Rare-Event Classification

From Credit Scoring to Financial Surveillance

Learning Objectives

- Explain why fraud detection is a **rare-event classification** problem, and connect it to the credit-scoring framework from Chapter 05
- Apply the base rate fallacy, Type I/II cost asymmetry, and cost-sensitive threshold selection to transaction data
- Implement and compare supervised, unsupervised, and hybrid fraud detection pipelines in Python
- Use **temporal cross-validation** (walk-forward) and explain why shuffled CV overstates performance
- Critically evaluate blockchain transparency claims for fraud detection

In Week 5, what was the base rate for loan defaults? How does ~1% fraud change the problem?

Answer: Loan defaults run at 10–25%. At 1% fraud, the base rate problem is far worse: even a good model generates mostly false alarms because legitimate transactions vastly outnumber fraud. Every metric behaves differently in this regime.

Agenda

Part I : Context: where fraud detection sits (blockchain, banking, and beyond)

Part II : The rare-event problem: base rate, accuracy trap, and proper metrics

Part III : Building a fraud detector on synthetic transaction data

Part IV : Tuning the decision: cost-sensitive thresholds and temporal validation

Part V : Unsupervised and hybrid methods: when labels are scarce

Part VI : Lab preview and CW2 scaffold release

Part I: Context

From credit scoring to fraud detection

In Chapter 05 you built a credit-scoring pipeline on the German Credit dataset: logistic regression, stratified CV, class weighting, ROC curves. That pipeline used a 30% default rate.

Fraud detection is **the same statistical problem** with harder numbers:

Property	Credit scoring (Ch 05)	Fraud detection (today)
Positive rate	10-25% (moderate)	<1% (severe imbalance)
Type I cost	Reject good borrower (~£50)	Block legitimate transaction (~£20)
Type II cost	Approve bad borrower (~£500)	Miss fraud (~£1,000-5,000+)
Cost ratio	~10:1	~50-100:1
Validation	Stratified CV	Temporal CV (fraud patterns drift)

The **framework** transfers directly. The numbers just make everything harder.

What happens to precision when the positive rate drops from 25% to 1%?

Fraud detection across domains

Domain	Positive class	Base rate	Cost ratio (FN:FP)
Credit card fraud	Fraudulent transaction	0.1-0.5%	50-100×
Loan default (Ch 05)	Borrower defaults	10-25%	5-10×
Money laundering (AML)	Suspicious activity	<0.01%	1000×+
Blockchain exploits	Malicious transaction	Varies	Varies
Insurance claims	Fraudulent claim	5-10%	10-20×

Every row is a **binary classification** with **asymmetric costs** and **class imbalance**. The tools are the same; only the numbers change.

Which row in the table has the hardest base-rate problem? Which did we already solve?

Answer: AML (anti-money laundering) at <0.01% is the hardest. We already built the loan default pipeline in Chapter 05 (10–25% positive rate). Today's fraud problem sits between the two.

Blockchain: transparency and limits

What blockchain gives you:

- Public ledger: every transaction visible (Bitcoin, Ethereum)
- Immutable history: cannot alter past records
- Graph structure: inputs/outputs create traceable chains

What it does not give you:

- Identity: addresses are pseudonymous, not anonymous
- Reversal: fraudulent transactions cannot be undone (no chargebacks)
- Context: no merchant categories, no customer profiles, no phone metadata

Firms like Chainalysis exploit the graph structure to trace funds (Chainalysis 2023). But the **statistical methods** for scoring suspicious transactions are the same as in traditional banking: logistic regression, random forests, cost-sensitive thresholds.

Three ways to agree without a central authority

Every blockchain must answer: *how do unrelated strangers agree on what happened, without trusting any single party?*

Proof-of-Work (Bitcoin): validators ("miners") compete to solve a computationally hard puzzle. The winner adds the next block and earns a reward. Cheating costs real electricity and hardware, making attacks economically irrational at scale.

Proof-of-Stake (Ethereum post-2022): validators lock up ("stake") cryptocurrency as collateral. Dishonest behaviour triggers **slashing** — a portion of the stake is automatically destroyed. Risk of loss replaces computational waste.

Permissioned BFT (Hyperledger, enterprise chains): a fixed, known set of consortium members vote on each block using Byzantine Fault Tolerant protocols. BFT tolerates up to f dishonest members among $3f + 1$ total nodes — so 10 members can tolerate 3 acting maliciously and still reach correct agreement. No puzzles, no staking; trust is enforced by legal contract and identity.

Consensus and security: comparison

Property	Proof-of-Work (Bitcoin)	Proof-of-Stake (Ethereum)	Permissioned BFT (Hyperledger)
Who validates	Miners (hash power)	Validators (staked capital)	Known consortium members
Attack cost	Electricity + hardware	Stake destroyed (slashing)	Compromise f+1 of 3f+1 nodes
Finality	Probabilistic (~60 min)	Strong (~15 min)	Immediate (1 block)
Energy	~150 TWh/yr	~0.01% of PoW	Negligible

For fraud detection, **finality** is what matters: it determines how quickly a flagged transaction can be acted on. Exchanges wait 6 Bitcoin confirmations (~60 min) precisely because of probabilistic finality.

Why can't Bitcoin simply make blocks final immediately?

Answer: Immediate finality requires a known, bounded set of validators who can all vote.

Part II: The Rare-Event Problem

The accuracy trap

Fraud rate: ~1% (10 in 1,000 transactions)

A model that predicts **every transaction as legitimate**:

- **Accuracy**: 99%
- **Fraud caught**: 0

```
import numpy as np
np.random.seed(42)
n = 10_000
fraud_rate = 0.01
y_true = np.random.binomial(1, fraud_rate, n)
y_naive = np.zeros(n, dtype=int)
acc = (y_naive == y_true).mean()
print(f"Naive model: predict ALL legitimate")
print(f"  Accuracy:     {acc:.1%}")
print(f"  Fraud caught: {(y_naive[y_true == 1] == 1).sum()} / {y_true.sum()}
print(f"  Recall:       0.0%")
```

Type I/II errors and cost asymmetry

True state	Predict legitimate	Predict fraud
Legitimate	Correct (TN)	**Type I**: false alarm
Fraud	**Type II**: missed fraud	Correct (TP)

Costs are not equal:

- Type I (false alarm): customer friction, support call (~£10-50)
- Type II (missed fraud): full transaction loss (~£500-5,000+)

Typical cost ratio: Type II is 50-100× more costly than Type I.

The optimal decision threshold minimises **expected cost**, not accuracy. This is the same framework from Ch 05, with different numbers.

If a missed fraud (Type II) costs 100× more than a false alarm (Type I), should the decision threshold be lower or higher than 0.5?

Answer: Lower. A lower threshold flags more transactions as fraud, catching more true

The confusion matrix: four outcomes

Every prediction falls into one of four cells. All metrics are just different ways of counting them.

	Predicted: legitimate	**Predicted: fraud**
Actually legitimate	**TN** — True Negative: correctly left alone	**FP** — False Positive: false alarm, customer frustrated
Actually fraud	**FN** — False Negative: missed fraud, money lost	**TP** — True Positive: fraud correctly caught

Plain-English summary:

- **TP** (True Positive): the model said fraud and it was fraud. The win.
- **TN** (True Negative): the model said legitimate and it was. The quiet majority.
- **FP** (False Positive): the model cried wolf. A real customer blocked. Costly in friction.
- **FN** (False Negative): the model missed real fraud. The most expensive failure.

Metrics built from the confusion matrix

All three metrics answer a different question about your model's performance:

Precision — of everything you flagged, how much was real fraud?

$$\text{Precision} = \frac{\overbrace{TP}^{\text{fraud correctly flagged}}}{\underbrace{TP + FP}_{\text{everything you flagged}}}$$

The denominator is your alert list. The numerator is how many of those alerts were genuine. Low precision means your alert list is mostly noise: analysts spend most of their day investigating legitimate transactions.

Recall (sensitivity) — of all real fraud, how much did you catch?

$$\text{Recall} = \frac{\overbrace{TP}^{\text{fraud correctly flagged}}}{TP + FN}$$

Part III: Building a Fraud Detector

Synthetic transaction data

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import warnings
warnings.filterwarnings("ignore")

np.random.seed(42)
n = 50_000

days = np.sort(np.random.uniform(0, 730, n))
epoch = days / 730

amount = np.random.lognormal(3.5, 1.8, n).clip(1, 25_000)
hour = np.random.choice(24, n)
is_weekend = np.random.binomial(1, 2/7, n)
velocity_1h = np.random.exponential(1.5, n).clip(0, 20)
is_foreign = np.random.binomial(1, 0.08, n)
```

What each feature represents

Feature	What it measures	Why it matters for fraud
`amount`	Transaction value (£)	Unusually large amounts relative to history
`hour`	Hour of day (0–23)	Fraudsters prefer 1–5 am (low oversight)
`is_weekend`	Weekend flag (0/1)	Reduced bank monitoring at weekends
`velocity_1h`	Transactions on this card in past hour	Card-testing: rapid small transactions before a big one
`is_foreign`	Merchant outside home country (0/1)	High-risk channel; card-not-present fraud
`days_since_open`	Account age in days	New accounts have shorter fraud history
`amount_ratio`	This transaction ÷ 30-day average	Spike in spending relative to normal behaviour

Before the model: your prediction

! Which feature is the strongest fraud signal?

The logistic regression we are about to fit will rank all seven features by predictive power. **Before you see the result**, rank these four candidates from strongest to weakest:

1. **Transaction amount** — unusually large transaction
2. **Hour of day** — 1–5 am, low oversight window
3. **Card velocity** — burst of transactions on this card in the last hour
4. **Foreign merchant** — card used outside home country

Write your ranking down. We will compare against the model in two slides.

Supervised pipeline: stratified CV

Two key choices before we run the model:

- **Stratified CV** splits the data into 5 folds while preserving the fraud rate in each fold. Without stratification, some folds could end up with zero fraud cases, making evaluation meaningless.
- `class_weight='balanced'` tells the model to treat each fraud case as worth ~99 legitimate cases during training. Without this, the model learns to ignore the 1% minority entirely because doing so already minimises overall error.

```
from sklearn.model_selection import StratifiedKFold, cross_val_score
from sklearn.linear_model import LogisticRegression
from sklearn.preprocessing import StandardScaler
from sklearn.pipeline import Pipeline

X = txn.drop(columns=["is_fraud"])
y = txn["is_fraud"]

pipe = Pipeline([
```

Part IV: Tuning the Decision

The default threshold catches zero fraud

Logistic regression outputs a **probability** between 0 and 1 for every transaction. The **decision threshold** is the cut-off: any transaction with predicted fraud probability above the threshold is flagged. By default, scikit-learn uses 0.5. What happens if you use that standard 0.5 threshold on **unweighted** logistic regression?

```
from sklearn.model_selection import train_test_split
from sklearn.metrics import confusion_matrix

pipe_plain = Pipeline([
    ("scaler", StandardScaler()),
    ("clf", LogisticRegression(max_iter=1000, random_state=42)),
])

X_tr, X_te, y_tr, y_te = train_test_split(X, y, test_size=0.3,
                                          stratify=y, random_state=42)
pipe_plain.fit(X_tr, y_tr)
y_prob = pipe_plain.predict_proba(X_te)[:, 1]
```

Cost-sensitive threshold selection

The idea is simple: count your false alarms, multiply each by its cost; count your missed frauds, multiply each by its cost; add them together. The threshold that gives the lowest total is the one you deploy.

$$\text{Expected Cost} = \underbrace{n_{FP} \times C_{FP}}_{\text{false alarm cost}} + \underbrace{n_{FN} \times C_{FN}}_{\text{missed fraud cost}}$$

```
cost_fp, cost_fn = 20, 1000
thresholds = np.arange(0.005, 0.50, 0.005)
costs = []
for t in thresholds:
    y_pred = (y_prob >= t).astype(int)
    cm = confusion_matrix(y_te, y_pred)
    tn, fp, fn, tp = cm.ravel()
    costs.append((fp * cost_fp + fn * cost_fn) / len(y_te))

opt_idx = int(np.argmin(costs))
```

Temporal CV: shuffling is cheating

Fraud patterns **drift** over time (our synthetic data has this built in: early fraud is foreign-transaction-heavy; later fraud shifts toward rapid-fire velocity patterns). Shuffled CV lets the model see future fraud patterns while being "tested" on the past. That is look-ahead bias.

Correct approach: `TimeSeriesSplit` (expanding window, always train-on-past, test-on-future):

Fold	Train period	Test period
1	Months 1–6	Months 7–12
2	Months 1–12	Months 13–18
3	Months 1–18	Months 19–24

Each fold trains only on past data and tests on the immediate future. The training window grows; the model never sees the future. Contrast with shuffled CV, which mixes January and December in the same training fold.

The divide-by-4 rule

Logistic coefficients are hard to read raw because they live on a log-odds scale. Two things help:

- **Standardised coefficients**: the `StandardScaler` in our pipeline rescales every feature to mean 0, standard deviation 1. This puts all coefficients on the same scale, so bar lengths in the chart below are directly comparable.
- **Divide by 4**: the logistic (sigmoid) curve is steepest at $p = 0.5$, where its slope is exactly $1/4$. So $/4$ gives the **maximum** probability change a one-unit shift in that feature can produce. At a ~1% base rate the actual marginal effect is much smaller, but the ranking tells you which features the model relies on most.

```
pipe.fit(X, y)
coefs = pd.Series(pipe.named_steps["clf"].coef_[0], index=X.columns)
top5 = coefs.abs().nlargest(5).sort_values()

labels = {
    "hour": "Hour of day",
    "is_foreign": "Foreign merchant",
```

Part V: Real Blockchain Data (Elliptic)

The Elliptic Bitcoin dataset

Everything so far used synthetic data. Now the real thing: **203,769 Bitcoin transactions** labelled by Elliptic (a blockchain analytics firm) as illicit (ransomware, scams, darknet markets), licit (exchanges, wallet services), or unknown (Weber et al. 2019).

Property	Value
Labelled transactions	46,564 (4,545 illicit, 42,019 licit)
Unlabelled	~157,000 (realistic: most transactions lack ground truth)
Features	166 per transaction (94 local + 72 from graph neighbours)
Time steps	49 discrete periods spanning ~1 year
Edges	234,355 directed Bitcoin flows
Illicit rate (labelled)	~9.8%

Supervised pipeline on Elliptic

```
from pathlib import Path

ell_path = Path("data/elliptic/elliptic_labelled.parquet")
if not ell_path.is_file():
    print("Elliptic parquet not found. See data/elliptic/README.md for dow
else:
    ell = pd.read_parquet(ell_path)
    feat_cols = [c for c in ell.columns if c.startswith("feat_")]
    X_ell = ell[feat_cols].values
    y_ell = (ell["class"] == 1).astype(int).values
    ts = ell["time_step"].values

    pipe_ell = Pipeline([
        ("scaler", StandardScaler()),
        ("clf", LogisticRegression(class_weight="balanced", max_iter=1000,
    ])
```

Walk-forward validation on Elliptic

The Elliptic dataset has 49 time steps with **genuine temporal drift**. We train on all data up to time step t, then test on the next block. This is the gold-standard evaluation.

```
from sklearn.metrics import roc_auc_score

if ell_path.is_file():
    boundaries = [1, 10, 20, 30, 40, 50]
    wf_results = []
    for i in range(len(boundaries) - 2):
        train_mask = (ts >= boundaries[0]) & (ts < boundaries[i + 1])
        test_mask = (ts >= boundaries[i + 1]) & (ts < boundaries[i + 2])
        if train_mask.sum() < 50 or test_mask.sum() < 50:
            continue
        X_tr_e, y_tr_e = X_ell[train_mask], y_ell[train_mask]
        X_te_e, y_te_e = X_ell[test_mask], y_ell[test_mask]
        pipe_ell.fit(X_tr_e, y_tr_e)
        auc_wf = roc_auc_score(y_te_e, pipe_ell.predict_proba(X_te_e)[:, 1
        illicit_rate = y_te_e.mean()
```

Part VI: Unsupervised and Hybrid Methods

When labels are scarce

In practice, labelled fraud data is **expensive** (analyst investigation per case) and **incomplete** (novel fraud types are missing from the training set, as the Elliptic walk-forward just showed). Unsupervised methods learn **normal** behaviour and flag deviations.

Three approaches:

Method	Idea	Strength	Weakness
Isolation Forest	Anomalies are easy to isolate (few splits)	Fast, scales well	No label feedback
Autoencoder *(Week 10 preview)*	Neural network compresses data, then reconstructs it; anomalies	Captures non-linear patterns	Needs tuning, GPU

How Isolation Forest works

Core intuition: anomalies are *easy to isolate* (Liu, Ting, and Zhou 2008). A normal transaction looks like its neighbours; a fraudulent one is already in a sparse region and needs very few binary splits to separate it from the crowd. See Isolation Forest in the glossary.

The algorithm:

1. Draw a random feature, then a random split point within that feature's range
2. Keep splitting recursively until each observation is in its own leaf
3. **Anomaly score** = average path length across many trees (shorter path = more anomalous)

	Normal transaction	Anomalous transaction
Neighbours	Many (dense region)	Few (sparse region)
Splits to isolate	Many	**Very few**
Score (lower = worse)	Close to 0	**Negative**

Isolation Forest: anomaly scores

```
from sklearn.ensemble import IsolationForest

iso = IsolationForest(contamination=0.02, random_state=42, n_estimators=200
scores = iso.fit(X).score_samples(X)

fig, ax = plt.subplots(figsize=(9, 4))
bins = np.linspace(scores.min(), scores.max(), 50)
ax.hist(scores[y == 0], bins=bins, alpha=0.6, label="Legitimate", density=T
ax.hist(scores[y == 1], bins=bins, alpha=0.6, label="Fraud", density=True)
ax.set_xlabel("Isolation Forest anomaly score (lower = more anomalous)")
ax.set_ylabel("Density")
ax.set_title("Unsupervised anomaly scores vs true labels (synthetic data)"
ax.legend()
plt.tight_layout()
plt.show()

flagged = scores < np.percentile(scores, 2)
```

Hybrid detection: anomaly score as feature

The production pattern: feed the **unsupervised anomaly score** as an extra feature into the supervised model. Why does this help? Isolation Forest captures **multivariate weirdness**: combinations of features that are jointly unusual, even if each feature looks normal on its own. Logistic regression sees features independently (unless you manually add interactions), so the anomaly score gives it information it could not extract alone.

```
X_aug = X.copy()
X_aug["iso_score"] = scores

pipe_aug = Pipeline([
    ("scaler", StandardScaler()),
    ("clf", LogisticRegression(class_weight="balanced", max_iter=1000, ran
])

auc_base = cross_val_score(pipe, X, y, cv=cv, scoring="roc_auc").mean()
auc_aug = cross_val_score(pipe_aug, X_aug, y, cv=cv, scoring="roc_auc").mea

print(f"Supervised only:                AUC = {auc_base:.3f}")
```

Framework reuse: one toolkit, many domains

	Credit scoring (Ch 05)	**Synth. fraud (today)**	**Elliptic Bitcoin (today)**
Positive rate	10-25%	~1%	~10% (labelled)
Shuffled CV AUC	~0.78	~0.78	~0.97
Temporal CV AUC	N/A (cross-section)	~0.77	~0.90
Look-ahead gap	N/A	~0.01	~**0.065**
Key lesson	Basics	Cost-sensitive threshold	Temporal drift matters

Key lesson: statistical principles transfer across domains, but **honest evaluation on real temporal data** reveals how much shuffled CV overstates performance.

Looking at the comparison table: which dataset showed the biggest look-ahead bias gap, and why?

Answer: Elliptic, at +0.065 AUC. Because it has genuine temporal drift in fraud typologies — real fraud campaigns evolved over the year of data, so shuffled CV leaked

Blockchain-specific detection challenges

What the public ledger adds:

- **Graph analytics**: every transaction links input addresses to outputs, forming a directed graph. **Centrality** measures how connected an address is (high-centrality nodes route many flows, like hubs); **community detection** groups addresses that transact heavily with each other (fraud rings tend to cluster). Both reveal structure invisible in tabular data (Akoglu, Tong, and Koutra 2015)
- **Temporal patterns**: sudden address activation, rapid fund movement across chains
- **On-chain forensics**: firms like Chainalysis and Elliptic trace fund flows across addresses to link pseudonymous wallets to real entities (Chainalysis 2023)

What it complicates:

- **Mixing services** and **privacy coins** (Monero, Zcash) deliberately obscure the transaction graph
- **DeFi exploits** (flash loans, oracle manipulation) can happen atomically — within a single transaction block
- **Pseudonymity**: no customer features (age, income, credit history) unless KYC is enforced

DeFi exploits: flash loans and oracle manipulation

Three attack types that do not exist in traditional banking:

Flash loan attack Borrow millions in a single transaction block, manipulate a price or protocol, profit, repay — all atomically. If the repayment fails, the entire transaction reverts. No collateral required. Losses from a single exploit have exceeded $100M (Perez and Livshits 2021).

Oracle manipulation DeFi protocols read prices from on-chain "oracles". An attacker with large capital temporarily distorts a thinly-traded token's price (e.g. via a flash loan), triggering a misliquidation or arbitrage before the oracle corrects.

Re-entrancy A smart contract calls an external contract mid-execution; the external contract calls back before state updates are written — draining funds in a loop. The DAO hack (2016, $60M) was re-entrancy (Atzei, Bartoletti, and Cimoli 2017).

Why detection is hard: all three complete in a single block (~12 seconds on Ethereum). Traditional fraud monitoring is too slow; on-chain simulation before inclusion is the only viable defence.

Part VII: Lab Preview and CW2 Release

Lab 8: fraud detection pipeline

Part A: Synthetic transaction data (exercises 1-7)

1. **Accuracy trap**: naive model, 99% accuracy, zero recall
2. **Supervised pipeline**: logistic regression, stratified CV, AUC and AP
3. **Default threshold disaster**: 0.5 threshold catches no fraud
4. **Cost-sensitive threshold**: sweep thresholds, plot expected-cost curve
5. **Isolation Forest**: unsupervised anomaly scoring, overlay with true labels
6. **Hybrid model**: add anomaly score as feature, measure AUC lift
7. **Temporal CV vs shuffled**: quantify the look-ahead bias gap

Part B: Elliptic Bitcoin data (exercises 8-10)

8. **Supervised pipeline on real Bitcoin transactions**: AUC and AP
9. **Walk-forward validation**: train on past time steps, test on future; compare with shuffled CV
10. **Graph exploration** (extension): load the edge list, compute degree centrality, test whether high-centrality nodes are more likely to be illicit

CW2 scaffolds: released today

Scaffolds are on Blackboard now. Instructions in the CW2 brief.

Scaffold	Method	Data	Statistical core
A	Blockchain fraud detection	Elliptic Bitcoin (46K txns)	Rare-event classification, temporal CV, cost-sensitive thresholds
B	Tree-based factor investing	JKP + Bloomberg	Bias-variance, walk-forward, SHAP
C	Volatility forecasting	Bloomberg equity indices	GARCH, Mincer-Zarnowitz, temporal CV

What to do this week:

- Read the CW2 brief (Blackboard)
- Choose a scaffold that matches your interests
- Start the scaffold notebook (released alongside brief)

Key takeaways

1. Fraud detection is **rare-event classification**: the same framework from Ch 05 (credit scoring) transfers directly
2. **Accuracy is meaningless** for rare events: use precision, recall, AP, and PR curves
3. The **default 0.5 threshold catches zero fraud** when the base rate is ~1%: cost-sensitive thresholds are essential
4. **Temporal CV** prevents look-ahead bias: shuffled CV overstates performance, especially when fraud patterns drift
5. **Unsupervised methods** (Isolation Forest) detect novelty without labels; **hybrid** models combine both strengths
6. **Blockchain transparency** enables graph analytics but does not change the statistical fundamentals

References

Akoglu, Leman, Hanghang Tong, and Danai Koutra. 2015. "Graph-Based Anomaly Detection and Description: A Survey." *Data Mining and Knowledge Discovery* 29 (3): 626–88. https://doi.org/10.1007/s10618-014-0365-y.

Atzei, Nicola, Massimo Bartoletti, and Tiziana Cimoli. 2017. "A Survey of Attacks on Ethereum Smart Contracts (SoK)." In *Principles of Security and Trust (POST)*, 10204:164–86. Lecture Notes in Computer Science. Springer. https://doi.org/10.1007/978-3-662-54455-6_8.

Chainalysis. 2023. "Crypto Crime Report 2023." https://go.chainalysis.com/crypto-crime-report.html.

Goldstein, Markus, and Seiji Uchida. 2016. "A Comparative Evaluation of Unsupervised Anomaly Detection Algorithms for Multivariate Data." *PLOS ONE* 11 (4): e0152173. https://doi.org/10.1371/journal.pone.0152173.

Liu, Fei Tony, Kai Ming Ting, and Zhi-Hua Zhou. 2008. "Isolation Forest." In *Proceedings of the 8th IEEE International Conference on Data Mining (ICDM)*, 413–22. https://doi.org/10.1109/ICDM.2008.17.