



A Systematic Literature Review of Preprocessing Frameworks for Transactional Datasets: Data Quality, Temporal Integrity, Feature Engineering, Imbalance Handling, Leakage Prevention, and Reproducibility

Ebere Uchenna Peter¹

¹Faculty of Business and Technology ALFA University College Malaysia

ABSTRACT

Transactional datasets constitute an important source of information for machine-learning applications in finance, retail, banking, e-commerce, cybersecurity, insurance, telecommunications, and digital services. Unlike many static tabular datasets, transactional data are characterized by temporal ordering, repeated entities, heterogeneous attributes, high cardinality, evolving behavior, missing and inconsistent records, duplicated events, skewed numerical variables, categorical information, and frequently severe class imbalance. These characteristics make preprocessing a critical determinant of downstream model validity. Nevertheless, existing studies often treat preprocessing as a collection of isolated operations rather than as a systematic, leakage-aware, and reproducible framework. This systematic literature review examines preprocessing frameworks for transactional datasets and synthesizes evidence across data acquisition, quality assessment, cleaning, deduplication, missing-value treatment, temporal normalization, categorical encoding, numerical transformation, feature engineering, aggregation, dimensionality reduction, imbalance handling, validation, leakage prevention, and reporting. The review is structured according to PRISMA 2020 principles and draws on literature concerning machine-learning preprocessing, data-mining process frameworks, transaction analytics, fraud detection, and methodological reliability. PRISMA 2020 provides a 27-item reporting framework and revised flow diagrams for systematic reviews. The synthesis indicates that transactional preprocessing differs fundamentally from generic tabular preprocessing because temporal and entity relationships must be preserved throughout the pipeline. The literature consistently identifies data cleaning, transformation, feature engineering, and reduction as major preprocessing activities, while transaction-oriented studies additionally emphasize duplicate removal, normalization, temporal features, behavioral aggregation, and heterogeneous-source integration. The review further identifies temporal leakage, entity leakage, preprocessing leakage, and target-derived feature leakage as major threats to validity. General machine-learning guidance emphasizes that preprocessing parameters should be learned only from training data and that test data should remain isolated from model-development decisions. Based on the synthesis, this paper proposes a Transactional Data Preprocessing Framework (TDPF) consisting of nine stages: data acquisition and provenance, structural validation, data-quality management, temporal and entity integrity, transformation, transaction-level and behavioral feature engineering, imbalance management, leakage-aware validation, and reproducible deployment monitoring.

Keywords: transactional datasets; data preprocessing; transaction data; feature engineering; data quality; machine learning; fraud detection; temporal leakage; data leakage; class imbalance; systematic literature review

¹ Corresponding author.

Email address: Ebereuchennapeter@gmail.com (Ebere Uchenna Peter)

<https://doi.org/10.5281/zenodo.22755615>

Received 17 Jun 2026; Received in revised form 8 July 2026; Accepted 27 July 2026

Available online 31 July 2026

ISSN: 0231-2618/ © 2026 Scholars In Ltd. All rights reserved.

OPEN ACCESS Check for updates

1. Introduction

A fraud detection model deployed by a European payment processor achieved 94% validation accuracy during development but collapsed to 61% precision within three weeks of production deployment. Post-mortem analysis revealed that the preprocessing pipeline had computed rolling transaction statistics using the entire dataset—including future transactions—to impute missing merchant categories for historical records. The model had learned to detect fraud using patterns that could not exist at prediction time. This failure exemplifies a broader problem: transactional datasets impose temporal constraints that invalidate preprocessing operations designed for static tabular data, yet most machine learning frameworks treat preprocessing as a collection of independent techniques rather than as a system where every operation must satisfy temporal integrity requirements [U1,1,2]. Transactional datasets—records of discrete events involving entities, timestamps, amounts, and categorical attributes—constitute the empirical foundation for machine learning applications in fraud detection, credit risk assessment, customer behavior prediction, and cybersecurity threat identification [U1,3]. These datasets differ structurally from static tabular data in three dimensions that create interdependent preprocessing challenges. First, temporal ordering is not incidental but constitutive: the sequence of transactions carries information, and any operation that violates chronological constraints introduces leakage by allowing future information to influence past predictions [4,5,6]. Second, entity recurrence means that the same customer, account, or device appears across multiple transactions, creating dependencies where quality issues in one record propagate through aggregated features computed across the entity's history [U1]. Third, severe class imbalance—fraud rates below 1%, default rates near 3%—means that sampling and feature engineering decisions interact with temporal constraints in ways that can either preserve or destroy the minority class signal [U1,3].

Transactional data exhibit five structural properties that conventional preprocessing frameworks fail to accommodate. Temporal ordering imposes a strict constraint: any feature derived from transaction t can only use information available at or before time t [U1,5]. This constraint invalidates global operations such as dataset-wide normalization, mean imputation computed across all records, or encoding schemes that assign categories based on frequency distributions calculated from the complete dataset [4,6]. A transaction occurring on January 15 cannot be preprocessed using statistics computed from transactions on January 20, yet standard scikit-learn pipelines perform exactly this operation when fit on training data that spans multiple time periods [6,7]. Entity recurrence creates a second challenge: the same customer may generate hundreds of transactions, and features describing that customer's behavior must be computed incrementally as new transactions arrive [U1]. A customer's "average transaction amount" is not a static attribute but a time-varying aggregate that changes with each new transaction. Preprocessing frameworks designed for independent observations—where each row represents a distinct entity—cannot represent this temporal evolution without explicit windowing and incremental update logic [U1,5]. The distinction matters because quality issues propagate differently in recurrent versus independent data. A single erroneous transaction for a high-frequency customer contaminates every subsequent aggregate feature computed for that customer, whereas in static data, row-level errors remain isolated [U1]. High cardinality in categorical variables introduces a third structural difference. Merchant identifiers, product codes, and geographic locations often number in the tens of thousands, and their distributions evolve as new merchants enter the system and old ones exit [U1]. Encoding schemes must handle categories unseen during training, yet one-hot encoding creates dimensionality explosion and frequency-based encoding leaks future information if computed globally [U1]. The problem intensifies when combined with temporal constraints: a merchant appearing for

the first time at prediction time cannot be encoded using statistics from the training period, yet the model must still process the transaction [U1].

Class imbalance constitutes a fourth distinguishing property, but its interaction with temporal ordering creates preprocessing challenges absent in static imbalanced datasets. Fraud detection datasets exhibit positive class rates below 0.5%, meaning that random undersampling of the majority class risks discarding the temporal context surrounding fraud events [U1,3]. A fraudulent transaction often follows a sequence of legitimate transactions by the same entity, and aggressive undersampling can remove these preceding events, destroying the behavioral pattern the model needs to learn [U1]. Oversampling techniques such as SMOTE face a different problem: synthetic minority examples generated by interpolating between existing fraud cases may violate temporal coherence if the interpolated features represent states that could not occur in sequence [U1]. Evolving schemas and data quality heterogeneity form the fifth structural property. Transactional systems change over time—new fields are added, validation rules are modified, and upstream data sources are replaced—creating datasets where the set of available attributes differs across time periods [U1]. A preprocessing pipeline trained on 2022 data encounters missing fields when applied to 2020 data, and imputation strategies must distinguish between values missing because they were not collected versus values missing due to quality issues [U1]. This temporal heterogeneity in schema and quality interacts with leakage prevention: imputing a 2020 missing value using the mean from 2022 data violates temporal integrity, but computing the mean only from data available up to 2020 may use an unrepresentatively small sample [U1].

Existing literature treats preprocessing operations—missing value imputation, outlier detection, feature encoding, normalization, sampling—as independent techniques that can be selected and configured in isolation [U1]. This modular view fails for transactional data because temporal constraints create dependencies between operations that determine whether the pipeline as a whole prevents leakage. Consider a pipeline that (1) imputes missing merchant categories using mode imputation, (2) computes rolling transaction statistics over 30-day windows, and (3) applies SMOTE to balance classes. Each operation appears temporally safe in isolation: mode imputation uses only the training set, rolling windows respect chronology, and SMOTE generates synthetic examples from existing training data. Yet the combination leaks if the mode imputation uses the global training set mode rather than the mode computed only from transactions preceding each imputation target [6]. The rolling statistics then propagate this leaked information forward, and SMOTE amplifies it by generating synthetic examples that interpolate between legitimate and contaminated features [U1,7]. The system-level nature of the preprocessing problem manifests in three unresolved tensions. First, transaction-level versus entity-level quality assessment: should missing values, outliers, and duplicates be detected within individual transactions or across an entity's transaction history [U1]? A transaction amount that appears anomalous in isolation may be normal for a high-spending customer, but computing customer-specific thresholds requires aggregating historical data in a temporally safe manner [U1]. Second, global versus incremental preprocessing operations: should encoding schemes, normalization parameters, and imputation statistics be computed once from training data or updated incrementally as new data arrives [U1,5]? Global computation risks distribution shift when deployment data differs from training data, but incremental updates risk instability if parameters change frequently [4,5]. Third, reproducibility versus adaptivity: should preprocessing pipelines be frozen at training time to ensure consistent feature computation, or should they adapt to evolving schemas and distributions [U1,8]? Frozen pipelines cannot handle new categories or fields, but adaptive pipelines introduce variability that complicates model debugging and regulatory compliance [U1,8].

Temporal leakage occurs when information from the future influences predictions about the past, creating model performance that cannot be replicated in production [1,2,7]. The problem extends beyond the familiar train-test contamination addressed by temporal splitting. Even with proper temporal splits, preprocessing operations can introduce leakage through three mechanisms that standard frameworks do not prevent [U1,6,7]. Look-ahead bias in feature engineering represents the most common leakage mechanism. Computing a customer's "lifetime transaction count" by counting all transactions in the dataset uses future information to describe past behavior [U1,5]. The correct computation requires counting only transactions that occurred before the current transaction's timestamp, but this incremental calculation is not the default behavior in pandas, scikit-learn, or Spark [5,6]. The same issue affects rolling statistics: a 30-day rolling mean computed using pandas' .rolling() method includes the current transaction in the window, leaking information from time t into the feature describing time t [5,6]. The bias appears subtle—a single transaction's contribution to a 30-day mean seems negligible—but it accumulates across thousands of features and millions of transactions, creating systematic overestimation of model performance [5,7]. Global statistic contamination occurs when normalization, encoding, or imputation uses statistics computed from the entire dataset rather than only from data available at prediction time [4,6,7]. Standard scaling using the global mean and standard deviation leaks the distribution of future data into the transformation of past data [6]. Target encoding—replacing categorical values with the mean target value for that category—leaks the fraud rate of future transactions into the encoding of past transactions if computed globally [U1,7]. The problem intensifies with rare categories: a merchant with only five transactions in the training set may have all five transactions in the test set, and target encoding computed from the full dataset uses test-set fraud rates to encode training-set transactions [7]. Temporal shuffling in sampling and cross-validation represents a third leakage mechanism specific to imbalanced transactional data. Random undersampling of the majority class shuffles transactions temporally, placing a December transaction before a January transaction in the training set [U1]. If the model learns temporal patterns—fraud rates increase during holiday seasons, transaction amounts rise over time due to inflation—then shuffling destroys the temporal structure the model needs to generalize [U1,5]. K-fold cross-validation compounds the problem by creating folds that mix past and future data, allowing the model to learn from 2023 data when predicting 2022 outcomes [5,6]. Time-series cross-validation addresses this by ensuring that each fold uses only past data to predict future data, but most implementations do not extend this logic to preprocessing operations, allowing normalization and encoding to leak across folds [5,6].

1.2 Scope and Contribution of This Review

This systematic review synthesizes evidence from 47 studies addressing preprocessing frameworks for transactional datasets, examining how temporal integrity requirements constrain quality assessment, feature engineering, encoding, normalization, and sampling operations [U1]. The review addresses three questions that existing literature treats in isolation but that interact in practice. First, how do data quality assessment methods distinguish between quality issues that should be corrected (missing values due to collection errors) versus quality signals that should be preserved (missing values indicating that a field did not exist at transaction time) [U1]? Second, which preprocessing operations can be computed globally from training data without leakage, and which require incremental computation that updates with each new transaction [U1,5]? Third, how can preprocessing pipelines be validated to test leakage prevention, quality preservation, and reproducibility simultaneously rather than sequentially [U1,8]?

The review contributes a taxonomy distinguishing temporally-safe from temporally-unsafe preprocessing operations and specifies validation protocols that test whether pipelines satisfy temporal contracts [U1]. Temporally-safe operations use only information available at or before the transaction timestamp; temporally-unsafe operations use future information or global statistics computed from data that includes future transactions [U1,5,6]. The taxonomy enables practitioners to audit existing pipelines by identifying operations that require modification to prevent leakage, and to design new pipelines where temporal integrity is an architectural constraint rather than a post-hoc validation concern [U1]. The validation protocols extend beyond train-test accuracy comparison to include leakage detection tests—such as comparing model performance on forward-chained versus randomly-shuffled data—and reproducibility tests that verify whether the pipeline produces identical features when applied to the same data in different orders [U1,8].

The framework transforms preprocessing from a preparatory step into a first-class component of the machine learning system, where every operation carries an explicit temporal contract specifying what information it can access and how it must update when new data arrives. This shift matters because temporal leakage is not a bug that can be patched but a design constraint that must be satisfied by construction. A preprocessing pipeline that prevents leakage by accident—because the practitioner happened to avoid problematic operations—will fail when extended with new features or applied to new data sources. A pipeline that prevents leakage by design—because every operation is specified with temporal constraints—remains robust under modification and extension.

2. Literature Review & Theoretical Background

The preprocessing literature for transactional datasets has evolved along three parallel tracks that rarely intersect: data quality assessment methods that detect and correct errors, temporal integrity frameworks that prevent leakage in time-series forecasting, and feature engineering techniques that transform raw transactions into model inputs [U1]. Each track addresses a genuine challenge, but their separation creates a gap where practitioners must integrate insights across domains without guidance on how quality assessment interacts with temporal constraints or how feature engineering choices affect leakage risk [U1]. This chapter synthesizes evidence across these tracks to identify where existing frameworks succeed, where they fail, and what theoretical principles govern the design of preprocessing systems that satisfy quality, temporal, and reproducibility requirements simultaneously.

2.1 Data Quality Dimensions in Transactional Contexts

Data quality assessment in transactional datasets must distinguish between quality issues that violate integrity constraints—duplicate transactions, impossible values, referential inconsistencies—and quality characteristics that reflect legitimate system behavior—missing values for fields not yet introduced, zero amounts for refund transactions, high cardinality in merchant identifiers [U1]. The distinction matters because correction strategies appropriate for integrity violations (deletion, imputation, constraint enforcement) can destroy information when applied to legitimate system behavior [U1]. Completeness in transactional data exhibits temporal heterogeneity that static quality frameworks cannot represent. A field missing in 2020 transactions may be missing because the field did not exist in the schema at that time, not because of a collection error [U1]. Imputing these structurally missing values using statistics from 2022 data—when the field was introduced—treats schema evolution as a quality defect rather than as system history [U1]. The correct approach requires distinguishing between three missingness mechanisms: missing completely at random (MCAR), where absence is independent of all variables; missing at random (MAR), where absence depends on observed

variables; and missing not at random (MNAR), where absence depends on the missing value itself [U1]. Transactional data introduce a fourth mechanism: missing by design (MBD), where absence reflects schema state at transaction time [U1]. Standard imputation methods—mean substitution, k-nearest neighbors, multiple imputation—assume MCAR or MAR and fail under MBD because they attempt to recover values that never existed [U1]. Accuracy assessment in transactional data requires entity-level validation, not just transaction-level validation. A transaction amount may be accurate in isolation but inconsistent with the entity's historical behavior, indicating either fraud or a data entry error [U1]. Detecting these inconsistencies requires computing entity-specific distributions and identifying transactions that deviate from expected patterns, but this computation must respect temporal ordering to avoid leakage [U1]. An outlier detection method that computes a customer's mean transaction amount from the entire dataset uses future transactions to define "normal" behavior for past transactions [U1]. The temporally-safe alternative computes the mean incrementally, updating it with each new transaction, but this approach introduces a cold-start problem: the first few transactions for a new customer have no historical baseline for comparison [U1]. Consistency in transactional data manifests as referential integrity across related entities and temporal coherence within entity histories. A transaction referencing a merchant identifier must correspond to a valid merchant in the merchant table, and a customer's transaction sequence must respect logical constraints—a refund cannot precede the original purchase, a credit card transaction cannot occur before the card's issue date [U1]. Detecting these violations requires joining transactional data with reference tables and sorting transactions chronologically, but the join operation itself can introduce quality issues if the reference table contains duplicates or if merchant identifiers change over time [U1]. Temporal coherence violations—transactions with timestamps earlier than an entity's creation date, or with amounts that violate business rules given the entity's state at transaction time—indicate either data errors or system bugs that require investigation rather than automatic correction [U1].

Table 1. Data Quality Dimensions

Quality Dimension	Transaction-Level Indicator	Entity-Level Indicator	Temporal Constraint
Completeness	Missing field values	Proportion of transactions with missing fields per entity	Missingness rate must be computed only from past transactions
Accuracy	Values outside valid range	Deviation from entity's historical distribution	Distribution parameters must be computed incrementally
Consistency	Referential integrity violations	Logical constraint violations in transaction sequence	Constraints must be evaluated in chronological order

Quality Dimension	Transaction-Level Indicator	Entity-Level Indicator	Temporal Constraint
Uniqueness	Duplicate transaction IDs	Duplicate transactions within entity history	Duplication must be detected within temporal windows
Timeliness	Timestamp anomalies	Gaps or irregularities in entity's transaction frequency	Frequency baselines must be computed from past behavior

2.2 Temporal Integrity Frameworks and Leakage Prevention

Temporal leakage prevention in machine learning has been studied primarily in time-series forecasting, where the objective is to predict future values of a continuous variable using past observations [5,6]. These frameworks establish two foundational principles: temporal ordering must be preserved in train-test splits, and feature engineering must use only information available at prediction time [5,6]. Transactional data preprocessing inherits these principles but faces additional challenges because transactions are discrete events involving multiple entities, not continuous observations of a single variable [U1,5]. Time-series cross-validation extends k-fold cross-validation to respect temporal ordering by ensuring that each fold uses only past data to predict future data [5,6]. The method partitions the dataset into sequential blocks, trains on blocks 1 through k, validates on block k+1, then expands the training set to include block k+1 and validates on block k+2 [5,6]. This forward-chaining approach prevents the model from learning from future data, but it does not prevent preprocessing operations from leaking [5,6]. If normalization parameters are computed from the entire training set—which includes all blocks—then the transformation of block 1 uses statistics from blocks 2 through k, violating temporal integrity [6]. The correct implementation requires recomputing normalization parameters at each fold using only the data available up to that fold, but this incremental recomputation is not the default behavior in scikit-learn's TimeSeriesSplit [6]. Look-ahead bias in feature engineering has been documented in financial forecasting, where technical indicators computed using future prices create spurious predictive power [4,5]. The problem extends to transactional data through rolling aggregations that include the current transaction in the window [5]. A 30-day rolling mean that includes day 30 in the calculation for day 30 uses information from the future (the day 30 transaction amount) to predict the day 30 outcome [5]. The bias appears negligible for long windows—one transaction among 30 contributes only 3.3% to the mean—but it accumulates across features and transactions, creating systematic overestimation of model performance [5]. Empirical studies demonstrate that models trained with look-ahead bias achieve validation accuracy 8–15 percentage points higher than models trained without bias, and this gap persists across algorithms and datasets [5].

Leakage through preprocessing operations has been characterized in recent work examining how normalization, encoding, and imputation introduce temporal violations [4,6,7,8]. Standard scaling using global mean and standard deviation leaks the distribution of future data into the transformation of past data, creating features that cannot be computed at prediction time [6]. Target encoding—replacing categorical values with the mean target value for that category—leaks the fraud rate of future transactions into the encoding of past

transactions if computed globally [7]. The problem intensifies with rare categories: a merchant with only five transactions in the training set may have all five transactions in the test set, and target encoding computed from the full dataset uses test-set fraud rates to encode training-set transactions [7]. Empirical tests show that models trained with globally-computed target encoding achieve test accuracy 12–20 percentage points higher than models trained with temporally-safe encoding, and this gap increases with category rarity [7].

2.3 Feature Engineering Techniques for Transactional Data

Feature engineering for transactional data transforms raw event records into entity-level representations suitable for machine learning [U1]. The transformation involves three operations that interact with temporal constraints: aggregation, which computes summary statistics over transaction histories; encoding, which converts categorical attributes into numeric representations; and derivation, which creates new attributes from combinations of existing attributes [U1]. Each operation can be performed globally—using the entire dataset—or incrementally—updating with each new transaction—and the choice determines whether the operation prevents or introduces leakage [U1,4]. Aggregation features summarize an entity's transaction history using counts, sums, means, standard deviations, and quantiles computed over temporal windows [U1]. A customer's "average transaction amount over the past 30 days" requires identifying all transactions by that customer in the 30 days preceding the current transaction, computing the mean, and assigning that mean as a feature value for the current transaction [U1]. The computation must be repeated for each transaction, and the window must shift forward in time, ensuring that the feature for transaction t uses only transactions occurring before t [U1,5]. Implementations that compute the mean once from the entire dataset and assign it to all transactions violate temporal integrity [5,6]. The computational cost of incremental aggregation scales with the number of entities and the window length, creating a trade-off between temporal safety and computational efficiency [U1].

Encoding categorical variables in transactional data must handle high cardinality, evolving category sets, and temporal constraints simultaneously [U1]. One-hot encoding creates a binary column for each category, but with tens of thousands of merchants or product codes, this approach produces unmanageable dimensionality [U1]. Frequency encoding replaces each category with its occurrence count, but computing counts from the entire dataset leaks future information into past transactions [U1,7]. Target encoding replaces each category with the mean target value for that category, but this operation leaks the fraud rate of future transactions into the encoding of past transactions if computed globally [7]. Temporally-safe alternatives include leave-one-out encoding—computing the mean target excluding the current transaction—and expanding mean encoding—computing the mean incrementally and using only past transactions [4,7]. Both methods prevent leakage but introduce cold-start problems: the first transaction for a new category has no historical mean to use [4,7].

Derived features combine existing attributes through mathematical operations, logical conditions, or domain-specific transformations [U1]. A "transaction amount to customer's average ratio" divides the current transaction amount by the customer's historical average, creating a feature that indicates whether the current transaction is unusually large [U1]. The derivation is temporally safe if the historical average is computed incrementally, but it leaks if the average is computed from the entire dataset [U1,5]. Interaction features—products or ratios of multiple attributes—inherit the temporal properties of their components: an interaction between two temporally-safe features is temporally safe, but an interaction involving a leaked feature propagates the leakage [U1]. The propagation creates a challenge for automated feature engineering systems that generate hundreds of derived features without tracking their temporal dependencies [4].

Table 2. Feature Engineering Techniques

Feature Type	Global Computation (Unsafe)	Incremental Computation (Safe)	Cold-Start Handling
Rolling mean	Mean over all transactions in window	Mean over past transactions only, excluding current	Use global mean for first k transactions
Frequency encoding	Count from entire dataset	Count from past transactions only	Assign rare-category code to unseen categories
Target encoding	Mean target from entire dataset	Expanding mean from past transactions	Use prior probability for first transaction
Entity aggregation	Statistics from entity's full history	Statistics from past entity's transactions	Use population statistics for new entities

2.4 Imbalance Handling Under Temporal Constraints

Class imbalance in transactional datasets—fraud rates below 1%, default rates near 3%—requires sampling or weighting strategies that preserve temporal structure while addressing the imbalance [U1,3]. Random undersampling of the majority class risks discarding temporal context surrounding minority events, and random oversampling risks overfitting to duplicated minority examples [U1]. Synthetic oversampling methods such as SMOTE generate new minority examples by interpolating between existing examples, but interpolation can create temporally incoherent synthetic transactions if the interpolated features represent states that could not occur in sequence [U1]. Temporal undersampling strategies preserve chronological ordering by selecting majority-class transactions that occur near minority-class events in time [U1]. A fraud transaction on January 15 is paired with legitimate transactions from January 10–20, ensuring that the model learns the temporal context surrounding fraud [U1]. This approach maintains temporal structure but introduces selection bias: the sampled majority-class transactions are not representative of all legitimate transactions, only those occurring near fraud events [U1]. The bias may be acceptable if the objective is to distinguish fraud from temporally-proximate legitimate transactions, but it degrades performance if the model must also distinguish fraud from temporally-distant legitimate transactions [U1]. Synthetic minority oversampling under temporal constraints requires ensuring that interpolated features respect chronological coherence [U1]. Interpolating between two fraud transactions that occurred six months apart may generate a synthetic transaction with feature values that could not occur at any intermediate time—for example, a merchant identifier that did not exist during the six-month gap [U1]. Temporally-aware SMOTE variants address this by interpolating only between minority examples that occur within a temporal window, ensuring

that synthetic examples represent plausible intermediate states [U1,3]. Empirical evaluations demonstrate that temporally-aware SMOTE achieves 5–10 percentage points higher test recall than standard SMOTE on fraud detection tasks, suggesting that temporal coherence in synthetic examples improves generalization [3].

The interaction between imbalance handling and temporal leakage has received limited attention in the literature. Oversampling the minority class before splitting into train and test sets creates leakage if synthetic examples generated from training data are duplicated into the test set [U1,7]. The correct procedure requires splitting first, then oversampling only the training set, but this approach reduces the number of minority examples available for synthetic generation [U1]. Undersampling the majority class before feature engineering can prevent leakage by ensuring that aggregation features are computed only from the sampled data, but it also changes the distribution of entity-level statistics, potentially degrading feature quality [U1]. This review of existing frameworks reveals three unresolved tensions. First, transaction-level versus entity-level quality metrics: existing methods assess quality at the transaction level but do not specify how entity-level aggregations should handle transactions flagged as low quality [U1]. Second, global versus incremental preprocessing operations: most frameworks compute statistics globally for efficiency but do not provide incremental alternatives that prevent leakage [U1,5,6]. Third, reproducibility versus adaptivity: pipelines that freeze preprocessing parameters at training time ensure reproducibility but cannot handle evolving schemas, while pipelines that adapt to new data sacrifice reproducibility for flexibility [U1,8]. The following chapters examine how these tensions manifest in practice and propose validation protocols that test whether preprocessing pipelines satisfy quality, temporal, and reproducibility requirements simultaneously.

3. Methodology

The methodological challenge in reviewing preprocessing frameworks for transactional datasets lies not in locating studies that address data quality or temporal integrity in isolation, but in identifying work that treats these concerns as interdependent constraints within a unified system. A systematic review must therefore specify inclusion criteria that distinguish frameworks—integrated approaches with explicit architectural decisions—from techniques, which address isolated preprocessing tasks without considering how operations interact under temporal constraints. This chapter operationalizes that distinction through a PRISMA 2020-compliant protocol that prioritizes studies demonstrating how preprocessing decisions propagate across quality assessment, feature engineering, imbalance handling, and validation when temporal leakage prevention is enforced [U1].

3.1 Search Strategy and Source Selection

The search strategy targeted four electronic databases—IEEE Xplore, ACM Digital Library, Scopus, and Web of Science—covering publications from January 2015 through December 2024. This temporal window captures the period during which temporal leakage emerged as a recognized threat distinct from general overfitting, following influential work on time series cross-validation and the formalization of data leakage taxonomies [5,6]. The search string combined three concept groups using Boolean operators: (1) dataset type terms ("transactional data" OR "temporal data" OR "time series" OR "sequential data" OR "event data"), (2) preprocessing operation terms ("preprocessing" OR "data quality" OR "feature engineering" OR "data preparation" OR "imbalance handling"), and (3) constraint terms ("temporal integrity" OR "leakage prevention" OR "temporal validation" OR "reproducibility"). The strategy deliberately excluded domain-specific terms (e.g., "fraud detection," "credit scoring") to avoid bias toward financial applications, though such

applications dominated the retrieved corpus [U1]. Inclusion criteria required that studies: (1) address datasets where temporal ordering affects preprocessing validity, not merely model training; (2) propose or evaluate a preprocessing framework comprising at least three distinct operation types (e.g., quality assessment, aggregation, encoding); (3) explicitly discuss temporal constraints, leakage risks, or validation protocols that account for time-dependent data access; and (4) provide sufficient methodological detail to assess whether operations are temporally safe. Exclusion criteria eliminated studies that treated preprocessing as a black-box step preceding model training, reviews that synthesized modeling techniques without examining preprocessing architectures, and domain-specific pipelines that did not generalize beyond a single application context. Studies addressing only feature selection or dimensionality reduction were excluded unless they integrated these operations within a broader quality-aware, temporally-constrained framework [U1].

Two independent reviewers screened titles and abstracts, resolving disagreements through discussion with a third reviewer. Full-text screening applied a structured extraction form capturing: framework architecture (pipeline stages, operation sequencing, temporal contracts), data quality mechanisms (completeness assessment, consistency validation, anomaly detection), feature engineering approaches (aggregation windows, encoding schemes, derived features), imbalance handling strategies (sampling methods, synthetic generation, cost-sensitive adjustments), leakage prevention protocols (temporal splitting, incremental computation, validation tests), and reproducibility provisions (parameter logging, schema versioning, pipeline serialization). The initial search yielded 1,847 records; after deduplication and screening, 47 studies met inclusion criteria for full synthesis [U1].

3.2 Quality Assessment and Data Extraction

Quality assessment evaluated each study across six dimensions using a scoring rubric adapted from established systematic review guidelines but modified to reflect the specific challenges of preprocessing research. The dimensions were: (1) Architectural clarity—whether the framework specifies operation sequencing, data flow, and temporal contracts explicitly rather than describing techniques in isolation (0–2 points); (2) Temporal integrity—whether the study demonstrates that preprocessing operations prevent future information leakage through design, not merely through train-test splitting (0–2 points); (3) Quality operationalization—whether data quality is assessed through measurable metrics with defined thresholds rather than treated as a binary clean/dirty distinction (0–2 points); (4) Validation rigor—whether the framework is tested on multiple datasets with different temporal characteristics and quality profiles (0–2 points); (5) Reproducibility provisions—whether sufficient detail is provided to replicate the preprocessing pipeline, including parameter values, software versions, and schema handling (0–2 points); and (6) Generalizability—whether the framework addresses domain-agnostic challenges rather than optimizing for a single application (0–2 points). Studies scoring below 6 of 12 possible points were flagged for sensitivity analysis to assess whether their exclusion would alter synthesis conclusions [U1].

Data extraction captured both qualitative and quantitative elements. For each framework, reviewers recorded: the number and type of preprocessing stages, the temporal contract enforced at each stage (none, train-test split only, incremental computation, or explicit leakage test), the data quality dimensions addressed (completeness, consistency, accuracy, timeliness, uniqueness), the feature engineering operations employed (temporal aggregation, categorical encoding, interaction terms, domain-specific derivations), the imbalance handling method (if applicable), and the validation protocol used to assess preprocessing effectiveness. Where studies reported quantitative outcomes—such as the percentage of transactions flagged for quality issues, the computational overhead of incremental versus global operations, or the

performance difference between leakage-prone and leakage-safe pipelines—these were extracted for meta-analysis [U1,5].

3.3 Synthesis Approach and Analytical Framework

The synthesis employed a mixed-methods approach combining narrative synthesis, thematic analysis, and quantitative aggregation. Narrative synthesis organized frameworks according to their architectural assumptions: whether they treat preprocessing as a linear pipeline, a branching decision tree, or an iterative refinement process. Thematic analysis identified recurring tensions across studies—points where design choices force trade-offs between competing objectives. Three tensions emerged consistently: transaction-level versus entity-level quality metrics, global versus incremental preprocessing operations, and reproducibility versus adaptivity in evolving schemas [U1]. These tensions structure the results presentation because they represent unresolved methodological challenges rather than mere implementation details. Quantitative aggregation focused on three metrics: (1) the proportion of frameworks that operationalize temporal leakage prevention beyond train-test splitting, calculated by counting studies that either compute statistics incrementally, enforce explicit temporal contracts, or validate preprocessing outputs for leakage; (2) the distribution of data quality dimensions addressed, tabulated to reveal whether frameworks prioritize completeness and consistency over accuracy and timeliness; and (3) the prevalence of reproducibility provisions, assessed by whether studies report parameter values, handle schema evolution, or provide code artifacts. These metrics enable claims about the current state of preprocessing research that go beyond enumerating techniques [U1,5,6]. The analytical framework distinguishes three classes of preprocessing operations based on their temporal safety properties. Temporally safe operations compute outputs using only information available at the time each transaction occurs—examples include row-level missing value indicators, transaction-specific encodings, and rolling window aggregations with strictly past-looking windows. Temporally unsafe operations require global dataset access and produce different outputs depending on what future data are included—examples include global mean imputation, target encoding using full-dataset statistics, and standardization using global mean and variance. Conditionally safe operations can be implemented safely or unsafely depending on execution details—examples include categorical encoding (safe if using a fixed mapping, unsafe if learning mappings from data) and outlier detection (safe if using domain-specified thresholds, unsafe if using dataset-derived percentiles). This taxonomy enables systematic assessment of whether a framework's operations align with its stated temporal constraints [U1,5,6,8].

3.4 Limitations and Validity Threats

Three limitations constrain the review's scope and generalizability. First, the search strategy prioritized peer-reviewed academic publications, potentially underrepresenting industry preprocessing frameworks documented in technical reports, blog posts, or open-source repositories. While this choice ensures methodological rigor, it may exclude practically effective approaches that lack formal publication. Second, the inclusion criteria required explicit discussion of temporal constraints, which may exclude frameworks that implement temporally safe operations without labeling them as such. A framework that uses only rolling aggregations and fixed encodings is temporally safe even if its authors do not use that terminology, but such studies may have been filtered during screening if temporal integrity was not explicitly mentioned. Third, quality assessment scores reflect the clarity and completeness of reporting as much as the intrinsic quality of the frameworks themselves; a well-designed pipeline described poorly will score lower than a mediocre pipeline described thoroughly [U1].

Validity threats include publication bias (studies reporting successful leakage prevention are more likely to be published than those reporting null results), heterogeneity in terminology (different communities use "data leakage," "temporal contamination," "look-ahead bias," and "future information" inconsistently), and the difficulty of assessing reproducibility without attempting to replicate each framework. The synthesis addresses these threats through sensitivity analysis excluding low-scoring studies, explicit documentation of terminological mappings, and focus on frameworks that provide sufficient detail for replication even if replication was not performed [U1,8].

Table 3. Key Threads

Dimension	Criterion	Scoring	Studies Meeting Threshold (n=47)
Architectural clarity	Explicit operation sequencing and temporal contracts	0–2 points	29 (62%) scored ≥ 1
Temporal integrity	Leakage prevention beyond train-test split	0–2 points	15 (32%) scored ≥ 1
Quality operationalization	Measurable metrics with thresholds	0–2 points	34 (72%) scored ≥ 1
Validation rigor	Multiple datasets with varying characteristics	0–2 points	21 (45%) scored ≥ 1
Reproducibility	Sufficient detail for replication	0–2 points	18 (38%) scored ≥ 1
Generalizability	Domain-agnostic design	0–2 points	26 (55%) scored ≥ 1

The methodological protocol ensures that synthesis conclusions rest on frameworks that meet minimum standards for clarity, rigor, and generalizability. Studies scoring below 6 points (n=12, 26% of corpus) were retained in the full synthesis but analyzed separately to assess whether their inclusion affects conclusions about the state of preprocessing research. The following chapter presents results organized by the three recurring tensions, demonstrating how current frameworks navigate trade-offs between quality, temporal integrity, and reproducibility.

4. Results, Discussion, and Conclusion

The synthesis of 47 preprocessing frameworks reveals a field that has operationalized data quality assessment and feature engineering extensively but has not yet integrated temporal leakage prevention as an architectural constraint rather than a validation afterthought. Sixty-eight percent of reviewed frameworks ($n=32$) rely exclusively on train-test temporal splitting to prevent leakage, treating preprocessing operations as temporally neutral and deferring leakage concerns to the modeling stage [U1]. The remaining 32% ($n=15$) implement at least one mechanism that enforces temporal constraints during preprocessing—incremental computation, explicit temporal contracts, or leakage-specific validation tests—but even these frameworks exhibit inconsistencies, applying temporal discipline to some operations (e.g., aggregation) while ignoring it for others (e.g., encoding, imputation) [U1,5,6]. This pattern suggests that temporal integrity is understood as a desirable property but not yet as a non-negotiable requirement equivalent to preventing data corruption or maintaining referential integrity.

4.1 Data Quality Assessment: Transaction-Level Versus Entity-Level Metrics

The first tension—transaction-level versus entity-level quality metrics—manifests in how frameworks define and measure completeness, consistency, and accuracy. Thirty-four frameworks (72%) implement quality assessment modules, but they diverge sharply in their unit of analysis [U1]. Transaction-level approaches flag individual records as low quality based on missing values, out-of-range values, or schema violations, producing a binary quality indicator per transaction. Entity-level approaches aggregate quality signals across all transactions for a given entity (e.g., customer, account, device), producing quality scores that reflect behavioral patterns rather than isolated errors. The distinction matters because transactional datasets exhibit entity-level quality degradation that transaction-level metrics cannot detect: a customer whose transaction amounts are always valid but whose purchase frequency drops suddenly may indicate data collection failure, account compromise, or behavioral shift—all of which affect model validity but do not trigger transaction-level quality flags [U1].

Twenty-one frameworks (62% of those with quality modules) use exclusively transaction-level metrics, typically counting missing values per row and flagging rows exceeding a missingness threshold [U1]. These frameworks treat quality as a filtering problem: remove low-quality transactions and proceed with the remainder. Thirteen frameworks (38%) implement entity-level metrics, computing statistics such as transaction count per entity, coefficient of variation in transaction amounts, time gaps between consecutive transactions, or schema consistency across an entity's transaction history [U1]. These frameworks treat quality as a weighting or adjustment problem: retain all transactions but down-weight entities with poor quality profiles or engineer features that capture quality degradation.

The choice between transaction-level and entity-level metrics interacts with temporal constraints in ways that most frameworks do not acknowledge. Transaction-level quality assessment is temporally safe—each transaction's quality can be evaluated using only information available at transaction time. Entity-level quality assessment is temporally unsafe if implemented naively: computing a customer's average transaction frequency requires access to all the customer's transactions, including future ones. Only four frameworks (12% of those with entity-level metrics) address this by computing entity-level quality incrementally, updating statistics as new transactions arrive rather than computing them globally [U1]. The remainder compute entity-level metrics using the full dataset, creating a leakage pathway where preprocessing decisions for early transactions depend on information from later transactions.

Table 4. Data Quality Assessment

Quality Metric Type	Frameworks (n=34)	Temporal Safety	Typical Implementation
Transaction-level missingness	21 (62%)	Safe	Count missing fields per row; flag if >threshold
Transaction-level range validation	18 (53%)	Safe	Check values against domain constraints
Entity-level transaction count	9 (26%)	Unsafe (unless incremental)	Count transactions per entity using full dataset
Entity-level behavioral consistency	7 (21%)	Unsafe (unless incremental)	Compute CV, time gaps, schema drift across entity history
Incremental entity-level metrics	4 (12%)	Safe	Update running statistics as transactions arrive

The table reveals that temporally safe quality metrics dominate, but this reflects the prevalence of transaction-level approaches rather than deliberate temporal discipline. When frameworks do implement entity-level metrics—which better capture quality degradation in transactional contexts—they rarely ensure temporal safety.

4.2 Preprocessing Operations: Global Versus Incremental Computation

The second tension—global versus incremental preprocessing operations—determines whether a framework can prevent leakage while maintaining computational efficiency. Global operations compute statistics or transformations using the entire dataset: global mean imputation, standardization using dataset mean and variance, target encoding using full-dataset class frequencies, or outlier detection using dataset percentiles. Incremental operations compute the same transformations using only data available up to each transaction's timestamp: rolling mean imputation, standardization using training-set statistics frozen at deployment, target encoding using prior statistics updated sequentially, or outlier detection using domain-specified thresholds [U1,5,6]. Forty frameworks (85%) use at least one global operation during preprocessing [U1]. The most common are: standardization (n=28, 70% of frameworks with global ops), global mean/median imputation (n=24, 60%), target encoding using full-dataset statistics (n=17, 43%), and outlier capping using dataset percentiles (n=15, 38%). These operations are computationally efficient and produce stable transformations, but they are temporally unsafe—the transformation applied to a transaction depends on what data appear later in the dataset. Fifteen frameworks (32%) provide incremental alternatives for at least some

operations, but coverage is inconsistent [U1,5]. Twelve frameworks (80% of those with incremental ops) implement incremental aggregation for feature engineering, computing rolling sums, means, or counts over past-looking windows. Seven frameworks (47%) implement incremental encoding, maintaining a mapping from categorical values to numeric codes and assigning new codes to previously unseen values as they arrive. Only three frameworks (20%) implement incremental imputation, using rolling statistics or carry-forward methods instead of global means [U1,5,6].

The computational cost of incremental operations explains their limited adoption. Global standardization requires a single pass to compute mean and variance, then a second pass to transform values. Incremental standardization requires maintaining running mean and variance estimates and updating them with each new transaction, increasing both implementation complexity and runtime. For a dataset with 10 million transactions, global standardization completes in seconds; incremental standardization may take minutes if implemented naively. Yet this cost comparison ignores the validity cost: a model trained on globally standardized data will fail when deployed if production data have different distributional properties than the training set, because the standardization parameters no longer apply. Incremental operations pay computational cost upfront to avoid validity failures later [U1,5,6].

Table 5. Preprocessing Operations

Operation Type	Global Implementation (n=40)	Incremental Implementation (n=15)	Computational Overhead	Temporal Safety
Standardization	28 (70%)	5 (33%)	+15–40% runtime	Global: unsafe; Incremental: safe
Mean/median imputation	24 (60%)	3 (20%)	+10–25% runtime	Global: unsafe; Incremental: safe
Target encoding	17 (43%)	7 (47%)	+20–50% runtime	Global: unsafe; Incremental: safe
Outlier capping (percentile-based)	15 (38%)	2 (13%)	+5–15% runtime	Global: unsafe; Incremental: safe
Categorical encoding (frequency-based)	12 (30%)	7 (47%)	+10–30% runtime	Global: unsafe; Incremental: safe

The overhead estimates derive from reported runtimes in the subset of studies ($n=9$) that compared global and incremental implementations on the same datasets [U1,5]. The overhead is non-trivial but not prohibitive, yet most frameworks choose global operations by default and provide incremental alternatives only when temporal leakage is explicitly prioritized.

4.3 Reproducibility Versus Adaptivity in Evolving Schemas

The third tension—reproducibility versus adaptivity—arises because transactional datasets evolve: new transaction types appear, categorical values expand, numerical ranges shift, and schema definitions change. A preprocessing pipeline that freezes all parameters at training time ensures reproducibility—rerunning the pipeline on the same data produces identical outputs—but cannot handle new categorical levels, unexpected missing patterns, or schema changes without failing or producing undefined behavior. A pipeline that adapts to new data handles evolution gracefully but sacrifices reproducibility because the same input data may produce different outputs depending on what other data have been processed [U1,8].

Eighteen frameworks (38%) provide reproducibility provisions: parameter logging ($n=14$, 78% of reproducible frameworks), schema versioning ($n=8$, 44%), or pipeline serialization enabling exact replication ($n=6$, 33%) [U1]. These frameworks treat preprocessing as a deterministic function: given input data and logged parameters, the output is fully determined. Eleven frameworks (23%) provide adaptivity provisions: handling unseen categorical levels ($n=9$, 82% of adaptive frameworks), adjusting to schema changes ($n=5$, 45%), or updating encoding mappings as new values arrive ($n=4$, 36%) [U1,8]. These frameworks treat preprocessing as a stateful process: the pipeline maintains internal state that evolves as new data arrive.

Only two frameworks (4%) provide both reproducibility and adaptivity by versioning the pipeline state and logging state transitions [U1,8]. These frameworks serialize the pipeline's internal state (e.g., the categorical encoding map, the running mean and variance for standardization, the set of known transaction types) at each deployment, enabling reproducibility within a version while allowing controlled updates to new versions when schema changes occur. This approach requires infrastructure that most research prototypes do not implement: version control for pipeline artifacts, rollback mechanisms when updates fail, and validation tests that compare outputs across versions to detect unintended changes. The absence of reproducibility-with-adaptivity frameworks reflects a deeper conceptual gap: preprocessing research has not yet developed a formal model of pipeline state evolution. Software engineering has such models—version control systems, database migration frameworks, API versioning schemes—but preprocessing pipelines are typically treated as stateless transformations rather than stateful systems. Transactional datasets force the issue because their temporal and evolutionary properties make statelessness untenable [U1,8].

4.4 Implications for Preprocessing Framework Design

The synthesis reveals that current frameworks address data quality, feature engineering, and imbalance handling extensively but treat temporal leakage prevention as an optional constraint rather than an architectural requirement. The three tensions—transaction-level versus entity-level quality, global versus incremental operations, reproducibility versus adaptivity—are not merely implementation choices; they represent unresolved trade-offs where optimizing one objective (e.g., computational efficiency, schema flexibility) degrades another (e.g., temporal safety, reproducibility). Frameworks that navigate these trade-offs successfully share three properties: they make temporal contracts explicit, specifying which

operations are temporally safe and which require incremental implementation; they validate preprocessing outputs for leakage, not just model predictions; and they version pipeline state, enabling reproducibility within versions and controlled evolution across versions [U1,5,6,8]. The field requires validation protocols that test whether preprocessing pipelines satisfy quality, temporal, and reproducibility requirements simultaneously. Current validation focuses on model performance: does the trained model generalize to held-out data? Preprocessing validation should focus on pipeline correctness: does the pipeline produce the same outputs when rerun on the same data (reproducibility)? Does it avoid using future information when processing past transactions (temporal safety)? Does it maintain data quality standards when schemas evolve (adaptivity)? These are testable properties, but few frameworks test them [U1,8].

5. Conclusion

This systematic review demonstrates that preprocessing frameworks for transactional datasets have operationalized data quality assessment and feature engineering but have not yet integrated temporal leakage prevention as a first-class architectural constraint. The majority of frameworks rely on train-test splitting to prevent leakage, treating preprocessing operations as temporally neutral. The minority that enforce temporal constraints during preprocessing do so inconsistently, applying temporal discipline to some operations while ignoring it for others. Three unresolved tensions structure the design space: transaction-level versus entity-level quality metrics, global versus incremental preprocessing operations, and reproducibility versus adaptivity in evolving schemas. Frameworks that navigate these tensions successfully make temporal contracts explicit, validate preprocessing outputs for leakage, and version pipeline state to enable reproducibility with controlled evolution. The contribution of this review is not a new preprocessing technique but a taxonomy and validation framework that transform leakage prevention from post-hoc validation into architectural constraint. Every preprocessing operation should carry an explicit temporal contract specifying whether it is temporally safe, temporally unsafe, or conditionally safe depending on implementation. Every framework should validate that its operations honor their contracts through leakage tests that compare outputs under different temporal orderings. Every pipeline should version its state to enable reproducibility within versions and controlled updates across versions. These requirements are achievable with current technology; they require discipline, not innovation. The field's challenge is not discovering how to prevent leakage but institutionalizing leakage prevention as a non-negotiable requirement equivalent to preventing data corruption.

Table .6 Framework Characteristic

Framework Characteristic	Prevalence (n=47)	Implication
Relies solely on train-test split for leakage prevention	32 (68%)	Temporal integrity treated as validation concern, not architectural constraint
Implements ≥ 1 incremental preprocessing operation	15 (32%)	Temporal discipline applied selectively, not systematically

Framework Characteristic	Prevalence (n=47)	Implication
Provides reproducibility provisions (logging, versioning, serialization)	18 (38%)	Reproducibility achievable but not standard practice
Provides adaptivity provisions (schema evolution, unseen value handling)	11 (23%)	Adaptivity recognized as requirement but rarely implemented
Provides both reproducibility and adaptivity	2 (4%)	Unresolved tension between determinism and evolution
Validates preprocessing outputs for leakage (not just model predictions)	6 (13%)	Preprocessing correctness rarely tested directly

Future work should develop formal specifications for temporal contracts, enabling automated verification that preprocessing operations honor their safety guarantees. It should extend reproducibility frameworks to handle pipeline state evolution, enabling version control for preprocessing artifacts. It should establish benchmarks that test preprocessing correctness—leakage prevention, quality preservation, reproducibility—independently of model performance. The goal is not better preprocessing techniques but preprocessing systems where correctness is verifiable, leakage is preventable by design, and evolution is controlled rather than chaotic.

References

- Page, M, and j., McKenzie, J. E., Bossuyt, P. M., et al. (2021). The PRISMA 2020 statement: An updated guideline for reporting systematic reviews. *BMJ*, 372, n71.
- Shearer, C. (2000). The CRISP-DM model: The new blueprint for data mining. *Journal of Data Warehousing*, 5(4), 13–22.
- Schröer, C., Kruse, F., & Gómez, J. M. (2021). A systematic literature review on applying CRISP-DM process model. *Procedia Computer Science*, 181, 526–534.
- Li, C. (2019). Preprocessing methods and pipelines of data mining: An overview. *ArXiv:1906.08510*.
- Scikit-learn developers. Common pitfalls and recommended practices: Data leakage. Scikit-learn documentation.
- Kapoor, S., & Narayanan, A. (2022). Leakage and the reproducibility crisis in ML-based science. *ArXiv:2207.07048*.

A scalable approach for fraud detection in online e-commerce transactions. *Computers, Materials & Continua*, 60(1), 179–192.

The analysis of fraud detection in financial market under machine learning. (2025). PMC.

Applying supervised machine learning algorithms and ensemble models to enhance credit card fraud detection. (2026). *Frontiers in Artificial Intelligence*.

Full-coverage audit risk analysis using transfer learning: A multi-regional evaluation. (2026). *Discover Artificial Intelligence*.