Data Preprocessing & Feature Engineering

Data preprocessing is the critical foundation of any machine learning pipeline. Raw data is rarely ready for modeling — it contains errors, inconsistencies, missing values, and irrelevant features. Preprocessing transforms raw data into a clean, consistent, and optimized format that enables models to learn effectively.

This guide provides comprehensive, actionable guidance for handling datatype mismatches, missing values, outliers, cardinality issues, datetime features, and feature engineering techniques.

1. Datatype Mismatch

Datatype mismatches originate from how data is stored and represented, errors during data entry, modifications over time, or errors during extraction or integration from different sources. These issues must be identified and resolved before modeling.

1.1 Common Datatype Issues

- **String in integer/float features:** Numeric values stored as text (e.g., '1' instead of 1) or text representations of numbers (e.g., 'one' instead of 1). Prevents mathematical operations.
- **Float in integer features:** Integer values stored as floats (e.g., 1.0 instead of 1). May indicate precision loss or data storage issues.
- **Inconsistent boolean representations:** Different systems store booleans differently: True/False, T/F, 'Yes'/'No', 1/0, 'Y'/'N'. Standardization is required.
- **Float/integer in string features:** Features expected to contain text instead hold only numeric values. May be acceptable as strings, but converting to numeric often provides greater analytical flexibility.

1.2 Handling Datatype Mismatches

- **Replace with Null:** Datapoints with mismatches are replaced with null and later imputed using techniques discussed in Section 2. Use when mismatches contain junk/errors.
- **Drop Rows:** Rows containing datatype mismatches are removed. Use when mismatches reflect errors in the datapoint itself and only a few instances exist.
- **Regenerate and Upload Data:** If issues originate from extraction or merge errors, regenerate the data from the source and reload. Preferred solution when feasible.
- **Casting Datapoints:** Convert data to the desired datatype. Use when data is saved or marked as the wrong type but values are valid. Most common solution for systematic datatype issues.

Best Practice

Best practice workflow:

1. Identify mismatches using Feature Categorization or Datatype Mismatch tab
2. Investigate root cause (data entry error vs systematic storage issue)
3. If systematic: cast to correct type
4. If sporadic errors: replace with null or drop rows
5. If extraction error: regenerate from source

2. Missing Values (Nulls)

Nulls in data may originate from errors during generation/collection, integration/merging, transmission, changes in data collection over time, or removal of datapoints due to errors. Missing data requires careful analysis before choosing a handling strategy.

⚠ Warning

Always investigate WHY data is missing before choosing a solution. Missing data patterns can be:

- Missing Completely at Random (MCAR): No pattern — safe to drop or impute
- Missing at Random (MAR): Missing depends on other observed variables — impute carefully
- Missing Not at Random (MNAR): Missingness is informative — add flag or model separately

2.1 Missing Simultaneous Features

When specific datapoints are missing values for multiple features, there may be particular reasons that may or may not correlate with the target. Analyze the cause and correlation with the target before acting.

- **Add a Flag:** If missing rows show correlation with the target, add a binary flag feature: 1 where selected features are missing, 0 elsewhere. This captures the information that the data was missing.
- **Drop Rows:** Remove datapoints where multiple meaningful features are missing. Generally not recommended as it reduces data volume and may bias the model. Use only when missing data represents a special case and few instances exist.

2.2 Missing Features

If data is missing for a specific feature and cannot be easily obtained, several approaches are available. The choice depends on the percentage of missing values, feature importance, and data type.

📌 Note

Rule of thumb: The methods below (except drop feature/rows) are suitable when < 30% of values are missing. Beyond 30%, consider more complex approaches like predictive imputation models or dropping the feature entirely.

Add Flag: Create a binary indicator for missingness (1 = missing, 0 = present). Can be combined with other imputation methods to preserve information about which values were originally missing.

Drop Feature: Remove the feature entirely. Use when:

- Significant fraction (> 60%) of data is missing
- Feature provides little direct or indirect insight into target

- Feature is highly correlated with other complete features

Drop Rows: Remove datapoints where the feature has empty values. Use when:

- Very few instances have missing values (< 5%)
- Target distribution in missing rows matches overall distribution
- Missingness appears completely random

Imputation Methods: Various statistical imputation techniques are available:

Method	Feature Type	When to Use	Limitation
Mean	Continuous	Standard approach for symmetric distributions	Skewed by outliers
Median	Continuous	Robust to outliers and skew	Less efficient than mean for normally distributed data.
Mode	Categorical	Most frequent value	May not exist or be meaningful
Forward Fill	Time-series	Last observation carried forward	First row may remain null
Backward Fill	Time-series	Next observation carried backward	Last row may remain null
Custom Value	Any	Domain knowledge (e.g., 'never used')	Requires careful justification

Best Practice

Imputation best practices:

- Continuous features: Mean (symmetric) or Median (skewed)
- Categorical features: Mode
- Time-series: Forward fill or Backward fill
- Always compute imputation values from training set only, then apply to validation/test
- Consider adding a 'was_imputed' flag for important features

2.3 Missing Target Features

Missing targets cannot be handled like input features. Imputation introduces bias toward specific target values, distorting model learning.

Solutions for Missing Targets

- Drop rows with missing targets — standard approach
- Fetch, regenerate, and reupload data if feasible
- Do NOT impute target values — this creates artificial patterns

⚠ Warning

Never impute missing target values. This forces the model to learn patterns that don't exist, creating biased predictions. Always drop rows with missing targets or obtain the true values from the source.

3. Handling Outliers

Outliers are values that lie far from the rest of the data points in a given feature. They are categorized into valid and invalid outliers.

3.1 Types of Outliers

- **Invalid Outliers:** Present due to errors — data entry mistakes (typing 100,000 instead of 1,000), data corruption during generation/storage, or system bugs. These must be corrected or removed.
- **Valid Outliers:** Legitimate extreme values — e.g., annual income in billions, century-old buildings in real estate data, rare but real events. Not errors, but may still need handling depending on the model.

Business Context

Model sensitivity to outliers:

• High sensitivity: Linear regression, neural networks, K-Means, KNN, PCA, SVM
• Low sensitivity: Tree-based models (Decision Trees, Random Forest, Gradient Boosting)

If using outlier-sensitive models, handling outliers is critical for robust predictions.

3.2 Methods to Identify Outliers

- **Threshold Value:** Select a value based on domain knowledge. Values beyond this threshold are outliers.
  Example: zero as minimum threshold for Inventory (cannot be negative). Typically identifies invalid outliers.
- **Standard Deviation-Based Threshold:** Values beyond a specific multiple of standard deviation from the mean are outliers.
  Common threshold: ±3 standard deviations (captures 99.7% of data in normal distribution).
- **Inter-Quartile Range (IQR):** Values beyond specific multiples of IQR are outliers. IQR is the distance between 1st and 3rd quartiles (Q3 − Q1).
  Common threshold: Q1 − 1.5×IQR and Q3 + 1.5×IQR. Used when data doesn't follow normal distribution.

3.3 Handling Outliers

- **Drop Rows:** Remove datapoints containing outliers. Use for:
  - Invalid outliers that cannot be corrected from source
  - Only a few instances exist
  - Target distribution in outlier rows matches overall distribution

- **Replace with Null:** Replace outliers with null, then handle using imputation methods from Section 2. Also typically used for invalid outliers.

- **Clamping (Winsorizing):** Replace outlier values with a fixed threshold — typically the threshold used to identify them. Example: if threshold is 99th percentile, replace all values above it with the 99th percentile value. Preserves data volume while limiting outlier influence.

Best Practice

Decision framework:

1. Identify outliers using IQR (robust) or standard deviation (normal data)
2. Investigate: Are they errors or legitimate extreme values?
3. Invalid outliers: Drop or replace with null
4. Valid outliers + outlier-sensitive model: Use clamping
5. Valid outliers + tree-based model: Often no handling needed

4. Irregular Cardinality

Irregular cardinality refers to inconsistencies or unexpected variation in the number of unique values (cardinality) within a feature. Four types require attention.

4.1 Cardinality of 1

If a feature has only one unique value, it adds no information to the model. Investigate the data source to confirm this isn't due to generation/merge errors.

Solutions

- Reupload data: If cardinality of 1 results from errors and correct data exists
- Drop feature: If the feature genuinely has only one value

4.2 Low Cardinality for Continuous Features

Continuous features are expected to have high cardinality. Low cardinality implies:

- Values fall within a constrained range
- Data generation/merge error
- Unintended transformation (e.g., age grouped into young/adult/old instead of actual values)

Solutions

- Reupload data: If data issues caused low cardinality

- Replace with null: If contains errors
- Drop feature: If large number of datapoints are erroneous

4.3 High Cardinality for Categorical Features

Categorical features are expected to have low cardinality. High cardinality (typically > 50 unique values) warrants investigation. May indicate data quality issues or inconsistent encoding.

Solutions

- Reupload data: If data issues caused high cardinality
- Aggregate/Replace feature levels: Combine levels with same meaning (e.g., 'm', 'M', 'male' → 'male')
- Drop feature levels: Remove rows with meaningless or erroneous values (few instances)
- Replace with null: If specific values are erroneous and cannot be corrected

4.4 Extremely High Cardinality

Very high cardinality (close to total datapoint count) indicates the feature contains an identifier — serial numbers, transaction IDs, unique keys. These provide no insight to models but may contribute to overfitting.

Solution

Drop feature after examination. Identifiers should not be used as model inputs.

> ⚠ **Warning**
> Exception: Do NOT drop datetime identifiers (timestamps). These contain rich temporal information that should be extracted into meaningful features (year, month, day of week, etc.) — see Section 5.

5. DateTime Feature Handling

DateTime features (timestamps, dates, times) contain rich temporal information but cannot be used directly by most ML models. They require transformation into numerical or categorical features that capture temporal patterns.

5.1 Why Transform DateTime Features?

- Most ML algorithms require numerical inputs — raw timestamps ('2024-03-15 14:30:00') are strings
- Direct conversion to Unix timestamps loses interpretability and cyclical patterns
- Temporal patterns (seasonality, day-of-week effects, holidays) are critical for many business problems

- Proper extraction enables models to learn time-based behaviors

5.2 DateTime Component Extraction

Common temporal components to extract:

Component	Values	Use Case	Relevant For
Year	2024	Long-term trends; seasonal patterns	Most models
Month	1-12 or Jan-Dec	Seasonality; business cycles	Most models
Day of Month	1-31	Monthly patterns	Specific domains
Day of Week	0-6 or Mon-Sun	Weekly cycles (weekday/weekend)	Retail, web traffic
Hour	0-23	Intraday patterns	Energy, traffic, operations

5.3 When to Keep vs Drop DateTime Components

Always Extract

These components are useful in most time-aware models:

- **Year:** Long-term trends
- **Month:** Seasonal patterns (retail, agriculture, tourism)
- **Day of Week:** Weekly cycles (weekday/weekend behavior)
- **Is Weekend:** Simplified weekly pattern (binary)

Extract When Relevant

Domain-dependent components:

- **Hour:** Energy consumption, traffic patterns, web activity, call center volumes
- **Day of Month:** Monthly billing cycles, payroll patterns, subscription renewals
- **Quarter:** Financial reporting, business planning cycles
- **Week of Year:** Inventory management, operations planning
- **Is Holiday:** Retail sales, travel bookings, restaurant reservations (requires holiday calendar)

Rarely Useful

- **Minute/Second:** Only for high-frequency data (stock trading, sensor data, server logs)
- **Millisecond:** Ultra-high-frequency applications only

Business Context

Business Examples:

• **Retail Sales Forecasting:** Extract month (holiday seasons), day_of_week (weekend spikes), is_holiday
• **Energy Demand:** Extract hour (peak/off-peak), day_of_week, month (seasonal), is_weekend
• **Customer Churn:** Calculate days_since_last_purchase, days_until_contract_end, month (renewal patterns)
• **Fraud Detection:** Extract hour (unusual transaction times), day_of_week, is_holiday (atypical patterns)

5.4 Cyclical Feature Encoding

Some temporal components are cyclical — December (12) wraps back to January (1), hour 23 wraps to hour 0. Linear encoding loses this cyclical nature. Use sine/cosine transformation for cyclical features.

When to Use Cyclical Encoding

- Month: Seasonal patterns where Dec-Jan are similar
- Hour: 24-hour cycles where 23:00 and 00:00 are close
- Day of Week: If Sunday and Monday have similar patterns

Formula

For a feature with period P (e.g., 12 months, 24 hours, 7 days):

$$\text{sin\_component} = \sin(2\pi \times \text{value} / P)$$

$$\text{cos\_component} = \cos(2\pi \times \text{value} / P)$$

Note
Tree-based models (Decision Trees, Random Forest, Gradient Boosting) handle cyclical features naturally through splits, so cyclical encoding is less critical. However, linear models and neural networks benefit significantly from cyclical encoding.

5.5 Derived Temporal Features

Beyond extracting datetime components, create derived features that capture temporal relationships:

Elapsed Time

- Days since event: days_since_registration, days_since_last_purchase
- Time until event: days_until_contract_end, hours_until_deadline
- Duration: session_duration_minutes, project_length_days

Time-Based Aggregations

- Rolling averages: sales_last_7_days, avg_transactions_last_month
- Cumulative sums: total_orders_to_date, lifetime_value
- Rate of change: revenue_growth_rate, customer_acquisition_rate

Event Flags

- First occurrence: is_first_purchase, is_new_customer
- Recency: purchased_in_last_30_days
- Frequency: num_purchases_last_year

Best Practice

DateTime feature engineering checklist:

- Never use raw timestamps as model inputs
- Extract relevant components based on domain knowledge
- Add is_weekend for weekly patterns
- Add is_holiday for event-driven behavior (retail, travel)
- Use cyclical encoding for linear models and neural networks
- Create derived features: days_since, time_until, rolling aggregates
- 7. Drop original datetime column after extraction

5.6 When to Drop DateTime Features

Drop the original datetime column after extraction in these cases:

- All relevant temporal components have been extracted
- Derived features (elapsed time, aggregations) have been created
- No additional information remains in the raw timestamp

Keep the original datetime column (for reference, not modeling) if:

- You need to track exact timestamps for business logic or audit trails
- Future feature engineering may require recalculation
- Visualization or reporting requires exact dates

Warning

Do NOT include the original datetime column as a model feature after extraction. It creates data leakage and overfitting because the model memorizes specific timestamps rather than learning temporal patterns.

6. Feature Development

Feature development derives more relevant features from raw features, reducing model complexity, computational cost, and data requirements while increasing robustness and accuracy. These features are often derived from domain knowledge.

Business Context

Why feature engineering matters: Many ML algorithms struggle to automatically discover optimal feature combinations. For example, a model may learn that GDP and population individually predict quality of life, but GDP per capita (their ratio) is a far better predictor. Creating this feature explicitly:

- Reduces model complexity (fewer parameters to learn)
- Improves accuracy (captures the true relationship)
- Increases robustness (less sensitive to noise in individual features)
- Reduces data requirements (explicit relationships need less data to learn)

6.1 Feature Development Techniques

- **Ratio of Features:** Useful when the relationship between two quantities is relative rather than absolute. Ratios normalize across scale differences and highlight proportional relationships.
  Examples: Debt-to-income ratio, GDP per capita, price-to-earnings ratio, click-through rate

- **Product of Features:** Useful when the product of two or more features is more relevant than individual features.
  Examples: Revenue = Price × Quantity; Area = Length × Width; Total cost = Unit price × Units × Tax rate

- **Summation of Features**: Aggregates multiple related features to provide better insight.
  Examples: Total expenses (sum of rent + utilities + supplies); Total score (sum of test scores)

- **Difference of Features:** Useful when the difference or deviation is of higher significance.
  Examples: Profit = Revenue − Expenses; Temperature differential; Price change

- **Exponential of a Feature:** Useful when the effect of a variable grows non-linearly. Helps linear models approximate non-linear dynamics.
  Examples: Exponential growth models, compound interest, disease spread

- **Logarithmic of a Feature:** Common for handling skewed distributions, compressing wide-range values, or modeling diminishing returns. Log transformation improves linear separability and reduces outlier impact.
  Examples: Log(income), log(population), log(price) — compresses multiple orders of magnitude

- **Quotient of a Feature (Floor Division):** Similar to ratios but emphasizes integer multiples. Floor division (e.g., 5 // 2 = 2) is useful when fractions have no meaning.

Examples: Number of years from days count (days // 365), number of weeks from days (days // 7)

- **Remainder of a Feature (Modulo):** Captures periodic or cyclical patterns. Useful for time-series and behavioral modeling where cycles matter.
  Examples: Day of week from day count (days % 7), hour within day (minutes % 60)

Best Practice

Feature engineering best practices:

- Base new features on domain knowledge and business logic
- Test multiple variations (ratio, product, difference) and compare model performance
- Remove original features if derived features are more predictive
- Use feature importance analysis to validate engineered features add value

7. Feature Selection

Redundant features or features uncorrelated with the target increase overfitting risk. Identifying and removing these features before training improves model robustness and accuracy.

7.1 Statistical Measures

Correlation and covariance measure linear relationships between features. These are useful but imperfect — they don't capture non-linear relationships and can miss indirect connections through intermediate features.

Note

Limitation: If two features are linked via a third (e.g., GDP and GDP per capita are linked through population), they may appear uncorrelated despite being related.

7.2 Visualization

More robust than simple correlation, but also limited when features are linked indirectly.

- **Scatter Plot:** Visualizes relationships between two continuous features. If correlated, the plot follows a pattern; otherwise, points scatter randomly.
- **Small Multiple Bar Plot:** Visualizes relationships between two categorical features. Bar plots showing densities of one feature's levels are displayed for each level of the second feature. If correlated, bar plots look noticeably different across levels.
- **Stacked Bar Plot:** Alternative to small multiple bar plots. Used when one feature has few levels (typically ≤ 3).

- **Box Plot:** Visualizes relationships between categorical and continuous features. If correlated, the continuous feature's distribution differs noticeably across categorical levels.

7.3 Modeling-Based Selection

Builds models and analyzes input feature influence to identify redundant or non-correlated features. More robust than statistical/visual methods but computationally expensive.

Common Techniques

- **Forward Selection:** Start with no features; iteratively add most predictive features
- **Backward Elimination:** Start with all features; iteratively remove least predictive features
- **Boruta Algorithm:** Statistical test-based feature selection using shadow features
- **SHAP Feature Selection:** Game-theoretic approach to measure feature importance

Note

Modeling-based selection requires building multiple ML models with various feature combinations, resulting in high computational cost. Recommended only if the model suffers from significant overfitting and statistical/visual methods are insufficient.

8. Duplicate Datapoints

Duplicate datapoints may result from data storage or merging errors. They cause:

- Model bias due to over-representation of certain patterns
- Inflated performance metrics due to data leakage (same datapoint in train and test)

However, duplicates are sometimes intentionally created when certain situations require higher model accuracy. In such cases, ensure the same datapoint does not appear in both training and validation/test sets.

8.1 Identifying Duplicates

Duplicates are identified by comparing values across features (keys):

- **All features:** All values must match for datapoints to be duplicates
- **Selected features:** Only values in key columns must match (e.g., phone numbers, transaction IDs)

Best Practice

Best practice: After identifying duplicates, keep one instance and remove the rest. If duplicates have conflicting values in non-key columns, investigate and resolve manually.

9. Feature Encoding

Many ML algorithms require numerical inputs. String, boolean, and other non-numeric types must be converted. Even for algorithms accepting strings, numerical encoding often provides flexibility for operations and embedding domain knowledge.

9.1 One-Hot Encoding

Converts each categorical level into a new binary feature: 1 where that level appears, 0 elsewhere. Simplest and most common encoding technique.

Advantages

- No ordinal assumptions — treats all levels equally
- Works with all ML algorithms
- Clear interpretation

Limitations

- High cardinality creates many features (dimensionality explosion)
- Provides no information about relationships between levels

9.2 Level Encoding (Ordinal Encoding)

Assigns a numerical value to each feature level. Used when hierarchical relationships exist or with tree-based models.

When to Use

- Hierarchical relationships: small=0, medium=1, large=2
- Tree-based models: insensitive to numerical order assumptions

> ⚠ **Warning**
> Caution: Arbitrary numbering with linear models or neural networks may introduce unintended assumptions. For example, encoding colors as red=0, green=1, blue=2 implies red and blue are farther apart than red and green, which may be meaningless. Use one-hot encoding unless order is meaningful.

10. Feature Normalization

Varying feature scales and extreme values influence robustness, accuracy, and learning rates of many ML algorithms (regression, neural networks, KNN, SVM). Normalization brings feature values within a specified range.

Technique	Range	Strengths	Limitations
Max Absolute	[-1, 1]	Preserves distribution shape	Sensitive to outliers

Technique	Range	Strengths	Limitations
Min-Max	[0, 1] or [-1, 1]	Bounded range; intuitive	Very sensitive to outliers
Robust	Unbounded	Resistant to outliers (uses IQR)	Not strictly bounded
Z-score	Unbounded	Standard for normal distributions	Assumes approximate normality
Log	Positive values	Handles multiple orders of magnitude	Cannot handle negatives or zero
Custom	Variable	Domain knowledge embedded	Requires expertise

10.1 When to Normalize

- Always: Linear regression, logistic regression, neural networks, SVM, KNN
- Optional: Tree-based models (Decision Trees, Random Forest, Gradient Boosting) — scale-insensitive

Best Practice

Critical rule: Compute normalization parameters (mean, std, min, max) from training set ONLY. Then apply these same parameters to validation and test sets. Never fit normalization on test data — this causes data leakage.

11. Binning

Binning transforms continuous features into categorical ones by grouping values into bins and assigning ordered numbers.

11.1 Binning Techniques

- **Equal-Width Binning:** Bins created such that each interval has equal range. Standard approach for binning.
- **Equal-Frequency Binning:** Bins created such that each bin contains equal number of datapoints. Used when data distribution is highly skewed.
- **Custom Range Binning:** Bins created based on domain knowledge.
  - Example: Age ranges: 0-2 (infants), 2-18 (children), 18-60 (adults), 60+ (seniors)
  - Example: Income brackets: <$30K (low), $30K-$100K (middle), >$100K (high)

Best Practice

Use binning when:

• Business logic naturally groups values (size: S/M/L, income brackets)

• Continuous feature has non-linear relationship with target

Avoid binning when:

• Losing granularity harms predictive power

Best Practice for Data Preprocessing

Preprocessing is iterative. Start with basic cleaning, train a baseline model, analyze errors, then refine preprocessing based on what you learn. The goal is not perfection — it's reliable, robust data that enables your model to learn patterns effectively.