



Data Cleaning & Representation

Building the Foundation for Reliable Machine Learning

Data quality is the cornerstone of any successful machine learning project. Without clean, well-represented data, even the most sophisticated algorithms will produce unreliable results. Data cleaning systematically removes errors, inconsistencies, and noise from raw datasets, while proper representation ensures that algorithms can correctly interpret and process the information.

Eduardo Ogasawara
eduardo.ogasawara@cefet-rj.br
<https://eic.cefet-rj.br/~eogasawara>



Missing Values: Causes and Risks



Common Causes

Missing values emerge from data collection errors, incomplete survey responses, sensor failures, or genuinely unavailable information. Understanding the source is critical for choosing the right handling strategy.



Bias Risks

When missingness isn't random—for example, if high earners skip income questions—your analysis becomes biased. This systematic pattern can skew results and lead to incorrect conclusions.



Algorithm Failures

Many machine learning algorithms cannot process NA values and will either crash or silently produce incorrect results. Preprocessing is essential before model training.

Two primary strategies exist for handling missing data: **removal** (deleting rows or columns with NAs) and **imputation** (filling in missing values using statistical methods). Each approach has distinct trade-offs that must be evaluated based on your specific use case and the nature of the missingness.

Practice: NA Removal with DAL Toolbox

Implementation Example

Here's a practical demonstration using R and the classic iris dataset. We artificially introduce a missing value, then use the `na.omit()` function to remove any rows containing NAs.

```
# Load the iris dataset
iris <- datasets::iris
# Introduce a missing value
iris$Sepal.Length[2] <- NA
# Check original size
nrow(iris)
# 150
# Remove rows with NAs
iris.clean <- na.omit(iris)
# Verify cleaned size
nrow(iris.clean)
# 149
```



❏ **Quick Start:** Full implementation code and additional examples are available in the DAL Toolbox GitHub repository at https://github.com/cefet-rj-dal/daltoolbox/blob/main/transf/na_removal.md

Limitations of Removing NAs

Speed vs. Precision

NA removal is the fastest cleaning strategy—just a single line of code. It's perfect for rapid exploratory data analysis when you need quick insights. However, this convenience comes at a cost.

Dataset Reduction

Every row with any missing value gets deleted entirely, which can dramatically shrink your dataset. If missingness is common across columns, you might lose substantial amounts of valuable data.

Systematic Bias Risk

When missing data follows patterns—such as certain demographics being less likely to respond—deletion amplifies bias. Your remaining dataset no longer represents the original population accurately.

Use Case Recommendation

Best suited for exploratory analysis with small amounts of random missingness. For production models or systematic missing patterns, consider imputation methods instead to preserve data and avoid bias.

Categorical Variables: Why Encoding Matters



The Numeric Requirement

Most machine learning algorithms—from linear regression to neural networks—can only process numeric inputs. Raw categorical variables like "red," "blue," "green" must be converted to numbers before training.

The False Order Problem

Simply encoding categories as integers (red=1, blue=2, green=3) creates an artificial ordering. Algorithms will incorrectly assume blue is "greater than" red, introducing mathematical relationships that don't exist in the data.

One-hot encoding solves this by creating separate binary columns for each category. This approach is essential for linear models, support vector machines, and deep learning architectures where mathematical operations assume meaningful numeric relationships.



Raw Categories

Text labels like "cat," "dog," "bird"



One-Hot Encoding

Independent binary columns: cat=1/0, dog=1/0, bird=1/0



Algorithm Ready

No false ordering, mathematically valid inputs

References: Han, J., Kamber, M., & Pei, J. – Data Mining: Concepts and Techniques (3rd Ed.); Witten, I. H., Frank, E., Hall, M. A., & Pal, C. – Data Mining: Practical Machine Learning Tools and Techniques (4th Ed.)

Practice: Categorical Mapping

Using DAL Toolbox for Transformation

```
# Create categorical mapping object
cm <- categ_mapping("Species")
# Transform the iris dataset
iris_cm <- transform(cm, datasets::iris)
# View the results head(iris_cm)
```

The DAL Toolbox's `categ_mapping()` function provides a clean, reusable approach to categorical encoding. By creating a mapping object first, you can apply the same transformation consistently across training and test sets—a crucial step for model deployment.



📖 **Learn More:** Complete documentation and advanced examples available at https://github.com/cefet-rj-dal/daltoolbox/blob/main/transf/categorical_mapping.md

01

Create Mapping

Define which column to encode

02

Apply Transform

Convert categories to numeric representation

03

Verify Results

Inspect encoded columns for correctness

Comparing Encoding Strategies



One-Hot Encoding

How it works: Creates independent binary columns for each category, ensuring no false ordering.

Best for: Linear models, SVMs, neural networks. Safe default choice for most algorithms.

Drawback: High cardinality categories create many columns, increasing dimensionality.



Label Encoding

How it works: Maps categories to sequential integers (0, 1, 2, 3...).

Best for: Tree-based models (random forests, gradient boosting) that can handle ordinal relationships.

Drawback: Implies false ordering—risky for linear models and distance-based algorithms.



Target Encoding

How it works: Replaces categories with the mean of the target variable for that category.

Best for: High-cardinality features where one-hot would create too many columns.

Drawback: Risk of data leakage if not implemented carefully with cross-validation.

Your encoding choice should align with both your algorithm's requirements and your data's characteristics. Tree-based models can handle label encoding, while linear models require one-hot. Consider dimensionality, interpretability, and computational constraints when deciding.

Key Takeaways

Data cleaning ensures reliable inputs

Quality preprocessing is non-negotiable. Clean data is the foundation upon which accurate, trustworthy models are built. Never skip this critical step.

Categorical mapping avoids false order

Proper encoding prevents algorithms from learning incorrect relationships. One-hot encoding is the safe default for most scenarios.

NA removal is simple but limited

While fast and convenient for exploration, deletion can introduce bias and waste valuable information. Consider imputation for production systems.

Adapt preprocessing to the algorithm

Different algorithms have different requirements. Linear models need one-hot encoding; tree-based models can handle label encoding. Always match your preprocessing strategy to your modeling approach.