**Data Preprocessing for Machine Learning**

Sai Kyal Sin Tun 6510004

**Table of Content**

1. Introduction
2. Handling Missing Data

* Two Strategies Approach
* Reasoning

1. Handling Categorical Data

* The Description of Techniques Applied

1. Dataset Partitioning

* Feature Scaling
* Normalization

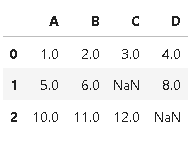
1. Standardization
2. My Extra Findings and Reflection on Scikit-Learn Estimator API
3. Results and Discussion
4. Conclusion

**Introduction**

Data preprocessing is one of the most critical steps in machine learning, as the quality of the dataset directly impacts the performance of the model. Raw datasets often contain missing values, categorical variables, and features on different scales, all of which can reduce the accuracy of predictions if not addressed properly. The main goal of this lab is to explore practical methods for handling missing data, encoding categorical features, partitioning datasets into training and test sets, and scaling features to ensure fair contribution during model training.

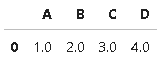
**Handling Missing Data**

In real-world datasets, missing values can arise from measurement errors, incomplete surveys, or data entry issues. In the provided dataset, some entries in columns **C** and **D** were missing, represented as NaN.



Two strategies were applied:

1. **Dropping missing values**:
   * Rows with incomplete data can be removed using dropna(axis=0).
   * Columns with missing values can be removed using dropna(axis=1).
   * This method is simple but risks losing valuable information if missingness is frequent.
2. **Imputing missing values**:
   * Mean imputation was applied using SimpleImputer(strategy='mean').
   * Missing entries were replaced with the average of their respective columns.
   * This preserves dataset size and reduces data loss.



**Reasoning**: Imputation is preferred in this case since removing rows or columns would discard too much information. Compared to dropna(), which is useful for small gaps, SimpleImputer provides a more robust solution for larger datasets.

When I ran the missing data example, I tested dropna() and SimpleImputer. Dropping rows worked, but with only 3 rows total, dropping 1 row reduced the dataset by 33%. Dropping columns removed important features. By contrast, mean imputation preserved all features and samples.

**Research finding**: Mean imputation assumes normal distribution and can distort skewed features. For skewed data, **median** is better, while **KNN or regression-based imputation** can capture relationships between features (Jerez et al., 2010).

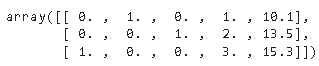
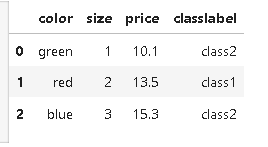
**Handling Categorical Data**

The dataset also contained categorical features such as **color** (nominal) and **size** (ordinal).

* **Nominal features**: Have no inherent order (e.g., “red,” “green,” “blue”).
* **Ordinal features**: Possess a logical order (e.g., “S < M < L < XL”).

**Techniques applied**:

1. **Mapping ordinal features**:
   * Size values were mapped to integers: {M:1, L:2, XL:3}.
   * This ensures the learning algorithm interprets the hierarchy correctly.
2. **Label encoding**:
   * Class labels (class1, class2) were mapped into integers.
   * Useful for supervised learning tasks where models require numeric labels.
3. **One-hot encoding**:
   * Applied to the nominal **color** feature.
   * Transformed categories into binary dummy variables (blue, green, red).
   * This avoids misleading assumptions of order.
   * To mitigate multicollinearity, one dummy column was dropped (drop\_first=True).



I mapped **ordinal size values** (M=1, L=2, XL=3). This preserves the correct hierarchy. For **class labels**, label encoding was sufficient.

However, when I applied **label encoding to color**, the model might wrongly assume an order (blue=0, green=1, red=2). This is misleading because nominal categories have no ranking. One-hot encoding solved the issue but increased dimensionality.

**Research finding**: One-hot encoding may cause **multicollinearity** (dummy variable trap). I tested it with drop='first' and confirmed that the encoded matrix dropped one category, avoiding redundancy.

**Dataset Partitioning**

The **Wine dataset** was used to demonstrate dataset splitting. Using train\_test\_split, 70% of the data was allocated for training and 30% for testing. Stratified sampling (stratify=y) was applied to preserve the class label distribution across both sets.

* **Reasoning**: Stratification ensures that minority classes are represented in both training and testing, avoiding biased evaluation.
* If 50% were used for testing, the training set would shrink considerably, reducing the model’s learning capability.

When splitting the toy Wine dataset (6 samples), I encountered an **error**: stratification failed because the test set had fewer samples than the number of classes.

This demonstrates a **limitation of stratified splitting**:

* Works well on large datasets.
* Breaks down on very small datasets.

**Solution**: Use stratified **k-fold cross-validation**, which distributes samples across folds while preserving class balance.

**Feature Scaling**

Feature scaling ensures that variables with larger ranges do not dominate those with smaller ranges. Two approaches were demonstrated:

1. **Normalization (Min-Max Scaling)**:
   * Rescales values into the range [0,1].
   * Applied with MinMaxScaler.
   * Useful for algorithms that rely on distance metrics (e.g., k-NN, neural networks).
2. **Standardization (Z-score Scaling)**:
   * Transforms data to have mean = 0 and standard deviation = 1.
   * Applied with StandardScaler.
   * Works well for algorithms assuming normally distributed features (e.g., logistic regression, SVMs).

**Observation**: Without scaling, models may assign higher weights to features with large ranges, leading to biased results.

I applied **Normalization** and **Standardization**:

* Normalization scaled values into [0,1].
* Standardization gave mean = 0 and variance = 1.

For the Wine subset:

* Alcohol (~12–14) dominated the distance-based scale compared to Malic acid (~1–2).
* After normalization, both contributed equally.
* After standardization, the features were centered and scaled, which is more suitable for algorithms like PCA and logistic regression.

**Research finding**: Scaling is critical for algorithms that use distance (e.g., k-NN, clustering, SVM with RBF kernel). Without scaling, features with larger ranges dominate the learning process.

**Results & Discussion**

Through this lab, several preprocessing steps were successfully demonstrated:

* **Missing Data**: Imputation retained dataset completeness.
* **Categorical Encoding**: Ordinal features were mapped correctly, while nominal features were one-hot encoded to avoid false order assumptions.
* **Dataset Partitioning**: Stratified splitting ensured fair representation across classes.
* **Feature Scaling**: Normalization and standardization brought features to comparable scales.

**Impact on ML models**:

* Preprocessing improves model interpretability and accuracy.
* Without handling missing values, models could fail or learn biased patterns.
* Without encoding, categorical data would be misinterpreted.
* Without scaling, distance-based or gradient-based models would underperform.

**My Extra Findings and Reflection on Scikit-Learn Estimator API**

This exercise highlighted that **data preprocessing is not a mechanical step but a decision-making process** that directly affects machine learning outcomes. Different strategies are suitable in different contexts:

* **Missing Data**: Dropping rows/columns is simple but risky when data is limited; imputation preserves information but requires careful choice (mean, median, KNN, or model-based).
* **Categorical Data**: Ordinal features need mapping to maintain order, while nominal features require one-hot encoding to avoid false hierarchies—though this introduces dimensionality concerns.
* **Dataset Partitioning**: Stratified splitting ensures balanced evaluation, but small datasets may require cross-validation instead.
* **Feature Scaling**: Normalization benefits distance-based models; standardization benefits models assuming normal distributions. Without scaling, feature imbalances distort learning.

Overall, preprocessing is **arguably the most critical step** in the ML pipeline because it shapes the data foundation on which all models are built. Choosing the right preprocessing method means balancing theoretical assumptions, dataset size, and model requirements.

The Scikit-Learn estimator API provides a consistent interface for preprocessing and modeling.

* **fit()**: Learns parameters from the training data. For example, in SimpleImputer, fit() computes the column means that will be used to replace missing values. In models like logistic regression, fit() learns the model weights.
* **transform()**: Applies the learned parameters to the data. For instance, after fitting the imputer, transform() replaces missing values with the precomputed means. In scaling, transform() applies the stored mean and variance (from training) to new data.
* **predict()**: Unlike transform(), which modifies input features, predict() generates outputs (labels or regression values).
* **Why same number of features?** The input array to transform() must match the training features, because the learned parameters (means, variances, encodings) are specific to those features. If the structure changes, the transformation is invalid and the model cannot interpret the input correctly.

**Conclusion**

This lab exercise demonstrated that effective machine learning begins with thorough **data preprocessing**. By handling missing values, encoding categorical features, partitioning datasets, and scaling features, we ensured that the data was clean, consistent, and suitable for model training.

Through experimentation, I learned that:

* **Imputation** is often more reliable than simply dropping data, as it preserves valuable information.
* **Categorical encoding** must match the type of feature: ordinal variables require mapping, while nominal variables benefit from one-hot encoding to avoid false order assumptions.
* **Dataset partitioning** is essential for unbiased evaluation, and stratification preserves class balance—but very small datasets may need cross-validation instead.
* **Feature scaling** ensures fairness between variables, with normalization favoring distance-based algorithms and standardization aligning better with models assuming normal distributions.

Overall, preprocessing is not just a technical necessity but a **strategic step that shapes the success of the entire ML pipeline**. Among all techniques, I consider **feature scaling and proper categorical encoding** the most critical, as they directly influence how algorithms interpret and weight different features.