Data Preprocessing & Model Tuning

Machine Learning Workshop – Day 2

Erdener Emin Eker

May 11, 2025

TEDU AI Data Science & AFIT Workshop

What We'll Cover Today

Main Topics:

- Cleaning data: missing values and outliers
- Feature engineering: encoding and transforming features
- Scaling: MinMax, Standard, and Robust scalers
- Model evaluation setup: train/test split and cross-validation
- Hyperparameter tuning: Grid Search and Random Search
- Building robust ML Pipelines

Goal: Prepare high-quality inputs for machine learning models and tune them effectively.

Why Preprocessing Matters

- Real-world datasets are noisy, incomplete, and inconsistent.
- ML models are sensitive to the structure and scale of input data.
- Poor preprocessing can lead to:
  - Biased or misleading predictions
  - Poor generalization to unseen data
  - Model failures in production
- Preprocessing is not a side task — it's core to ML success.

Dimensions of Data Quality

Validity: Ensures data conforms to defined rules and formats (e.g. correct type, range limits).

Accuracy: Measures how well data reflects true values (e.g. valid street address may still not exist).

Completeness: Are any required fields missing? Are we covering all necessary variables?

Consistency: No contradictions within or across datasets (e.g. a child marked as "married").

Uniformity: Standardization of formats and units (e.g. kg vs lbs, date formats).

Practical Steps in Data Cleaning

Step 1: Remove Duplicate or Irrelevant Observations Eliminate repeated rows and irrelevant entries that don't serve the analysis goal.

Step 2: Fix Structural Errors Correct typos, inconsistent capitalizations, and malformed categories (e.g. "N/A" vs "Not Applicable").

Step 3: Filter Unwanted Outliers Assess validity of extreme values — either treat, flag, or exclude them if inappropriate.

Step 4: Handle Missing Data Choose from dropping, imputing, or redesigning how the missing data is handled. Consider the reason for missingness.

Step 5: Validate and QA Ask: Does the data make sense? Does it support or contradict your assumptions?

A Real-World Dataset is Not a Clean CSV

Common Issues:

- Missing values and duplicates
- Mixed data types
- Outliers and rare categories
- Inconsistent encoding

color	director_name	duration	gross	movie_title	anguage	country	budget	title_year	imdb_score
Color	Martin Scorsese	240	116866727	The Wolf of Wall StreetÂ	English	USA	100000000	2013	8.2
Color	Shane Black	195	408992272	Iron Man 3Â	English	USA	200000000	2013	7.2
color	Quentin Tarantino	187	54116191	The Hateful EightÂ	English	USA	44000000	2015	7.9
Color	Kenneth Lonergan	186	46495	MargaretÂ	English	usa	14000000	2011	6.5
Color	Peter Jackson	186	258355354	The Hobbit: The Desolation of SmaugÂ	English	USA	225000000	2013	7.9
	N/A	183	330249062	Batman v Superman: Dawn of JusticeÂ	English	USA	250000000	202	6.9
Color	Peter Jackson	-50	303001229	The Hobbit: An Unexpected JourneyÂ	English	USA	180000000	2012	7.9
Color	Edward Hall	180		RestlessÂ	English	UK		2012	7.2
Color	Joss Whedon	173	623279547	The AvengersÂ	English	USA	220000000	2012	8.1
Color	Joss Whedon	173	623279547	The AvengersÂ	English	USA	220000000	2012	8.1
	Tom Tykwer	172	27098580	Cloud AtlasA	English	Germany	102000000	2012	-7.5
Color	Null	158	102515793	The Girl with the Dragon TattooÂ	English	USA	90000000	2011	7.8
Color	Christopher Spencer	170	59696176	Son of GodÂ	English	USA	22000000	2014	5.6
Color	Peter Jackson	164	255108370	The Hobbit: The Battle of the Five ArmiesÂ	English	New Zealand	250000000	2014	7.5
Color	Tom Hooper	158	148775460	Les MisÃ©rablesÂ	English	USA	61000000	2012	7.6
Color	Tom Hooper	158	148775460	Les MisÃ©rablesÂ	English	USA	61000000	2012	7.6

(Example: A sample messy dataset)

Types of Missing Data

Not all missing data is the same:

- **MCAR** – Missing Completely at Random → e.g., survey skipped by mistake
- **MAR** – Missing at Random → e.g., income missing, but depends on education
- **MNAR** – Missing Not at Random → e.g., people with high income skip income question

Why it matters: The type of missingness affects how we handle it and the bias introduced.

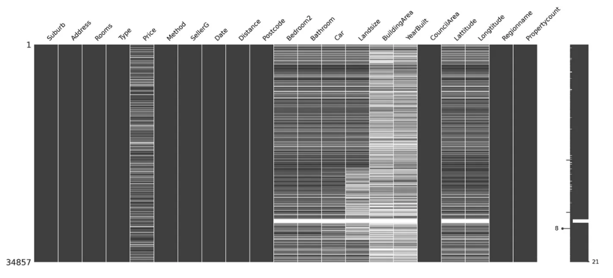
Strategies for Handling Missing Data

- **Remove:**
  - Drop rows/columns with missing values (if few)
- **Simple Imputation:**
  - Mean, median, or mode substitution
- **Advanced Imputation:**
  - KNN imputation
  - Regression imputation
  - Multivariate Imputation by Chained Equations (MICE)

Caveat: Imputation adds assumptions. Choose based on context and model sensitivity.

Visualizing Missing Values

- Detecting missing data is the first step in cleaning
- Useful tools in Python:
  - ‘df.isnull().sum()‘
  - ‘missingno.matrix(df)‘
  - ‘sns.heatmap(df.isnull())‘



(Visualizing missing patterns helps identify data issues)

Comparing Imputation Techniques

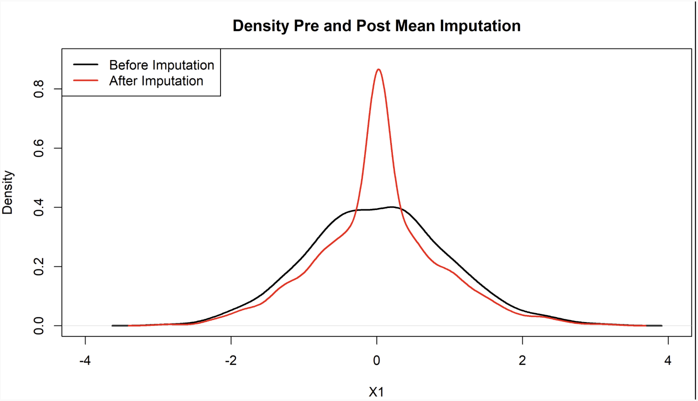
Each imputation method has trade-offs:

- **Mean Imputation:**
  - Fills missing values with the variable's mean.
  - Simple and fast, but sensitive to outliers.
  - Shrinks variance and can distort correlations.
- **Median Imputation:**
  - Uses the median value — more robust to outliers.
  - Preserves central tendency but still reduces variability.
  - Ideal when data is skewed.
- **K-Nearest Neighbors (KNN) Imputation:**
  - Estimates missing values based on similar observations.
  - Preserves multivariate relationships better.
  - Slower and more complex; sensitive to scaling.

Insight: Choose the method based on the missingness mechanism, variable distribution, and your model's sensitivity to distortions.

Impact on Distributions

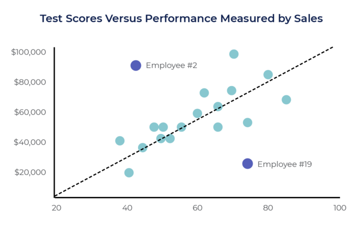
- Imputation isn't just about filling blanks
- It can significantly affect downstream model behavior
- Visualize before/after to ensure you're not distorting key signals



(Compare variable histogram before vs. after filling missing values)

What is an Outlier?

- An outlier is an observation that lies an abnormal distance from other values.
- Causes can include:
  - Measurement error
  - Data entry mistake
  - Rare, but valid event
- Outliers can distort:
  - Mean and standard deviation
  - Model coefficients (especially in linear models)
- Always inspect context before removal.

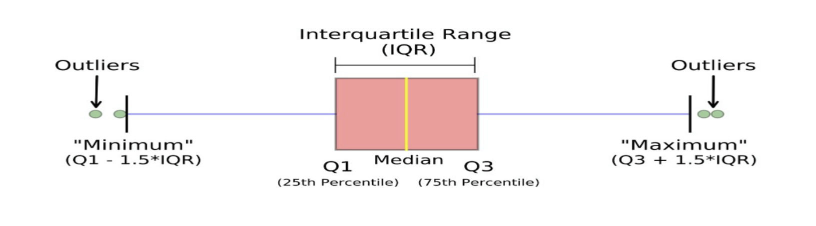


(Example: boxplot highlighting an outlier)

Outlier Detection Methods

Univariate methods:

- **Z-score:** Identifies values far from the mean. Rule of thumb: beyond 3 standard deviations.
- **Interquartile Range (IQR):** Flags values outside 1.5 $\times$ IQR from Q1 or Q3. Robust to non-normal data.



Tip: Use multiple methods to cross-check results.

Visual Outlier Detection

- Visualization helps identify patterns that statistics may miss.
- Common tools:
  - Boxplots – highlight extreme values in distributions
  - Scatterplots – detect bivariate anomalies
  - Histograms – reveal skewness or long tails
- Combine visual inspection with formal methods for a comprehensive approach.

Outlier handling should be tailored to your problem, not done automatically.

What is Feature Engineering?

- The process of creating, transforming, or selecting variables to improve model performance.
- Involves both domain knowledge and data understanding.
- Better features often outperform more complex models.
- Goals:
  - Improve predictive power
  - Make data more interpretable or structured
  - Handle raw or unstructured data effectively

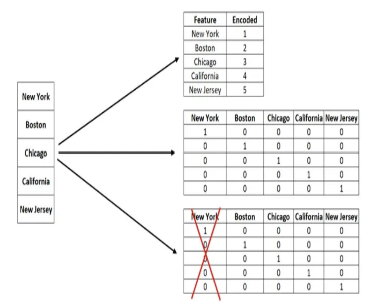
Encoding Categorical Variables

Why encode?

- Categorical features must be converted into numerical format.

Common Techniques:

- **Label Encoding:** Assigns a unique integer to each category. Implies ordinal relationship.
- **One-Hot Encoding:** Creates a new binary column for each category. No order is assumed.
- **Dummy Encoding:** Creates binary columns like One-Hot Encoding but drops one category to avoid multicollinearity. Typically used in linear models.



(Example: different encoding strategies for "Cities")

When to Use Which Encoding?

- **Tree-based models (e.g., Random Forest, XGBoost):**
  - Can handle label-encoded variables fairly well.
  - Prefer label or frequency encoding to reduce dimensionality.
- **Linear models (e.g., Logistic Regression, SVM):**
  - Require one-hot encoding for categorical variables.
  - Otherwise, model assumes incorrect linear relationship between categories.
- **High-cardinality categorical variables:**
  - One-hot encoding may lead to sparse, high-dimensional data.
  - Consider grouping infrequent categories or using frequency/target encoding.

Feature Transformation

- Transformations can help linearize relationships and stabilize variance.
- Common transformations:
  - **Log Transform:** Compresses long right tails.
  - **Square Root:** Handles moderate skewness.
  - **Box-Cox or Yeo-Johnson:** Data-driven, preserves normality.
  - **Polynomial Features:** Add non-linearity to linear models.
- Apply transformations with care to maintain interpretability.

Date and Time Feature Engineering

- Raw date variables are rarely useful in original form.
- Extract useful components:
    - Day, month, year, hour
    - Day of week, weekend flag
    - Time since event or rolling time windows
- Important for seasonality, trend detection, and time-aware models.
- Avoid leakage: do not extract future-based features in predictive models.

Why Scaling is Necessary

- Many machine learning models are sensitive to the scale of features.
- Unscaled features can dominate the learning process, especially in:
  - Distance-based models: k-NN, SVM
  - Gradient-based models: Logistic Regression, Neural Networks
- Scaling helps ensure that:
  - All features contribute equally
  - Optimization converges more smoothly
- Tree-based models (e.g., Random Forest, XGBoost) are generally scale-invariant.

Types of Feature Scaling Methods

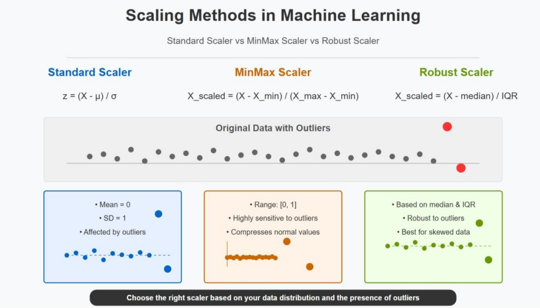
Standard techniques:

- **StandardScaler:**
  - Centers features around zero with unit variance.
  - Assumes approximately normal distribution.
- **MinMaxScaler:**
  - Scales features to a fixed range, usually [0, 1].
  - Sensitive to outliers.
- **RobustScaler:**
  - Uses median and IQR, making it robust to outliers.
  - Good for skewed distributions.

Choose the scaler based on your data distribution and model type.

Visualizing the Impact of Scaling

- Scaling changes the shape and range of data.
- Important to visualize before and after to detect unintended distortions.
- Helps debug downstream model performance issues.

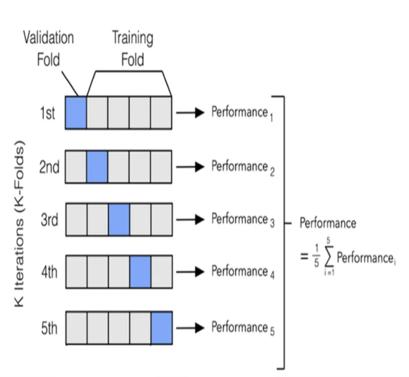


Train/Test Split

- Splitting data is essential to measure out-of-sample performance.
- Typical split: 70–80
- Important considerations:
  - **Stratification:** Ensures class balance in classification problems.
  - **Random seed:** Enables reproducibility.
  - **No leakage:** Do not scale or impute using test data.
- `train_test_split()` from `sklearn.model_selection` handles this efficiently.

Cross-Validation

- To evaluate model performance on multiple subsets.
- Common types:
  - **k-Fold CV:** Split into k equal folds; each used once as validation.
  - **Stratified k-Fold:** Maintains class balance in each fold.
  - **TimeSeriesSplit:** Maintains temporal order for forecasting.
- Reduces the risk of overfitting
- Essential for reliable model validation.



(Illustration: how data is split in k-Fold CV)

Common Pitfalls Without Cross-Validation

- Evaluating performance on a single test set can be misleading.
- Models might perform well on the split by chance (data leakage or overfitting).
- Hyperparameters tuned on a single test set may not generalize.
- Cross-validation provides a more stable estimate of model quality.
- Essential for small datasets where a single test split isn't representative.

Good models generalize — cross-validation helps prove it.

What is Hyperparameter Tuning?

- Hyperparameters are configuration settings not learned during model training.
- Examples:
    - Number of neighbors in k-NN
    - Maximum tree depth in decision trees
    - Regularization strength in linear models
- Tuning these values can significantly affect model performance.
- The goal is to find the combination that yields the best validation performance.

Grid Search vs Random Search

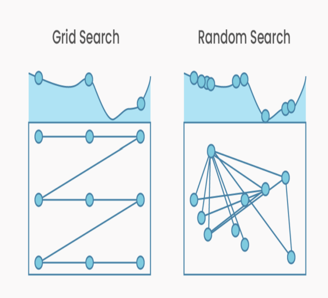
Grid Search:

- Exhaustive search over a defined parameter grid.
- Guarantees evaluation of all combinations.
- Can be computationally expensive.

Random Search:

- Samples parameter combinations randomly.
- More efficient with large hyperparameter spaces.
- Often finds good solutions faster than grid search.

Both require cross-validation to avoid overfitting to the validation set.



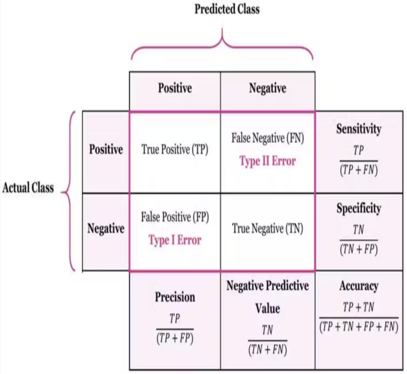
(Comparison of search coverage in parameter space)

Choosing the Right Evaluation Metric

- **Classification:**
  - Accuracy — simple but misleading on imbalanced data
  - Precision, Recall, F1 — for skewed classes
  - ROC-AUC — performance across thresholds
- **Regression:**
  - Mean Squared Error (MSE)
  - Mean Absolute Error (MAE)
  - R^2 Score

Tuning without the right metric can optimize the wrong outcome.

Always align your metric with the business or research goal.



(Confusion Matrix: TP, FP, FN, TN)

Why Use Pipelines?

- Pipelines combine preprocessing and modeling steps into a single object.
- Advantages:
  - Reduces code repetition and clutter
  - Prevents data leakage during preprocessing
  - Ensures consistent transformation across training and test sets
  - Enables easy deployment and cross-validation
- Provided by `sklearn.pipeline.Pipeline`

Think of a pipeline as an ML assembly line.

Typical Pipeline Structure

A simple example:

- `SimpleImputer()` → `StandardScaler()` → `LogisticRegression()`
- Encapsulated as:

```
Pipeline(steps=[
  ("impute", SimpleImputer()),
  ("scale", StandardScaler()),
  ("model", LogisticRegression())
])
```

- Pipelines can include encoding, scaling, feature selection, modeling, etc.
- For heterogeneous columns, use `ColumnTransformer`.

Cross-Validation and Tuning with Pipelines

- Pipelines integrate seamlessly with `GridSearchCV` and `RandomizedSearchCV`.
- Example:

```
param_grid = {
  "model__C":  [0.01, 0.1, 1, 10]
}
grid = GridSearchCV(pipeline, param_grid, cv=5)
```

- Hyperparameters can be tuned across all steps using double underscores.
- Allows reproducible and clean ML workflows from start to finish.

Summary of Best Practices

- Always inspect and handle missing values carefully.
- Detect and justify treatment of outliers — don't remove blindly.
- Choose encoding and scaling based on model type and data characteristics.
- Use train/test split and cross-validation to avoid overfitting.
- Tune models using appropriate metrics and CV strategies.
- Wrap preprocessing and modeling in pipelines to ensure reproducibility.

Good preprocessing and validation are the foundation of any reliable ML workflow.

→ Let's switch to Colab!