# Week 2- Feature engineering: Data encoding, scaling and Bias mitigation methods

Dr. Smitha K G
Senior Lecturer,
College of Computing and Data Science,
Nanyang technological University

NANYANG TECHNOLOGICAL UNIVERSITY SINGAPORE

# Data encoding

- Encoding categorical data is a process of converting categorical data into integer format so that the data with converted categorical values can be provided to the models to give and improve the predictions.
- Categorical data encoding is a fundamental step in preparing data for machine learning models, as most algorithms require numerical input.
- There can be two kinds of categorical data:
  - Nominal data (No intrinsic order data)
  - Ordinal data
- The choice of encoding technique depends heavily on the nature of the categorical variable (nominal or ordinal)

# Types of encoding

- Label/Ordinal Encoding (Ordinal)
- One-Hot Encoding (nominal)
- Count / Frequency encoding (nominal)
- Binary Encoding (nominal)
- Target Encoding (nominal)
- Hash Encoding (nominal)

# Label encoding

- This type of encoding is used when the variables in the data are ordinal. Ordinal encoding converts each label into unique integer values based on its rank.
- Only really useful if there exist a natural order in categories
  - Model will consider one category to be 'higher' or 'closer' to another
- Pros: Simple, memory-efficient, and effective for preserving the order of features.
- Cons: If the categories are not ordinal, the model will mistakenly assume an order and a linear relationship, which is often incorrect.

| ID | Product | Size | Color | Country | Category | Supplier | Price |
|---|---|---|---|---|---|---|---|
| 1 | Laptop | Small | Silver | USA | Premium | Dell | 1200 |
| 2 | Tablet | Medium | Black | China | Budget | Lenovo | 300 |
| 3 | Phone | Small | Blue | India | Premium | Samsung | 900 |
| 4 | Monitor | Large | Black | Japan | Standard | Sony | 400 |
| 5 | Laptop | Medium | Gray | Germany | Premium | HP | 1100 |
| 6 | Phone | Small | Silver | China | Budget | Xiaomi | 500 |
| 7 | Monitor | Large | White | USA | Standard | Dell | 450 |
| 8 | Tablet | Medium | Black | India | Budget | Samsung | 320 |
| 9 | Laptop | Large | Silver | Japan | Premium | Lenovo | 1250 |
| 10 | Phone | Small | Gray | Germany | Premium | Xiaomi | 850 |

In this dataset the best column for label or ordinal encoding -> Size

Why?

- It has a natural order: Small (1) < Medium(2) < Large(3)
- Ordinal encoding preserves this meaningful hierarchy
- Algorithms like trees or XGBoost work well with this

| ID | Size | Size_Ordinal |
|---|---|---|
| 1 | Small | 1 |
| 2 | Medium | 2 |
| 3 | Small | 1 |
| 4 | Large | 3 |
| 5 | Medium | 2 |
| 6 | Small | 1 |

# One Hot encoding

- In One-Hot Encoding (OHE)- for nominal data-each category of any categorical variable creates a new binary column for each unique category. After OHE, the number of one hot variables depends on the number of categories in the data
- Pros:
  - Creates independent features, prevents the model from assuming an artificial order (prevents misinterpretation), and is suitable for most linear (logistic regression, SVM) and distance-based models (K nearest Neighbors).
  - Provides clarity/transparency
  - OHE allows the model to learn the presence of each category, providing the most expressive and non-biased representation of nominal data
- Cons:
  - Curse of Dimensionality if a variable has many unique categories (high cardinality), increasing memory usage and computation time.
  - It also causes multicollinearity (as one column can be perfectly predicted by the others, known as the dummy variable trap).

| ID | Product | Size | Color | Country | Category | Supplier | Price |
|---|---|---|---|---|---|---|---|
| 1 | Laptop | Small | Silver | USA | Premium | Dell | 1200 |
| 2 | Tablet | Medium | Black | China | Budget | Lenovo | 300 |
| 3 | Phone | Small | Blue | India | Premium | Samsung | 900 |
| 4 | Monitor | Large | Black | Japan | Standard | Sony | 400 |
| 5 | Laptop | Medium | Gray | Germany | Premium | HP | 1100 |
| 6 | Phone | Small | Silver | China | Budget | Xiaomi | 500 |
| 7 | Monitor | Large | White | USA | Standard | Dell | 450 |
| 8 | Tablet | Medium | Black | India | Budget | Samsung | 320 |
| 9 | Laptop | Large | Silver | Japan | Premium | Lenovo | 1250 |
| 10 | Phone | Small | Gray | Germany | Premium | Xiaomi | 850 |

In this dataset the best column for one-hot encoding -> color

Why?

- Color is nominal (no order)
- Only 5 categories → small number
- One-hot avoids fake ordering
- Great for linear/logistic regression

| ID | Color | Color_Black | Color_Gray | Color_Blue | Color_Silver | Color_White |
|---|---|---|---|---|---|---|
| 1 | Silver | 0 | 0 | 0 | 1 | 0 |
| 2 | Black | 1 | 0 | 0 | 0 | 0 |
| 3 | Blue | 0 | 0 | 1 | 0 | 0 |
| 4 | Black | 1 | 0 | 0 | 0 | 0 |
| 5 | Gray | 0 | 1 | 0 | 0 | 0 |
| 6 | Silver | 0 | 0 | 0 | 1 | 0 |

# Count/Frequency encoding

- This type of encoding is used when the data is nominal.
- Count encoding technique replaces each category with the frequency (or count) of its occurrence in the dataset.
- Pros:
  - Does not increase the dimensionality of the dataset (dimensionality reduction).
  - Captures information about the prevalence of each category.(rare vs frequent)
  - Handles new or rare Categories: Rare or infrequent categories are assigned a small, distinct value, grouping them naturally and reducing their potential to cause overfitting. Unseen categories in the test set can be assigned a default frequency (e.g., 0) found in the training data.
- Cons: If two different categories have the same frequency, the model will treat them identically, leading to a loss of information.

| ID | Product | Size | Color | Country | Category | Supplier | Price |
|---|---|---|---|---|---|---|---|
| 1 | Laptop | Small | Silver | USA | Premium | Dell | 1200 |
| 2 | Tablet | Medium | Black | China | Budget | Lenovo | 300 |
| 3 | Phone | Small | Blue | India | Premium | Samsung | 900 |
| 4 | Monitor | Large | Black | Japan | Standard | Sony | 400 |
| 5 | Laptop | Medium | Gray | Germany | Premium | HP | 1100 |
| 6 | Phone | Small | Silver | China | Budget | Xiaomi | 500 |
| 7 | Monitor | Large | White | USA | Standard | Dell | 450 |
| 8 | Tablet | Medium | Black | India | Budget | Samsung | 320 |
| 9 | Laptop | Large | Silver | Japan | Premium | Lenovo | 1250 |
| 10 | Phone | Small | Gray | Germany | Premium | Xiaomi | 850 |

| ID | Color | Count_color | Frequency_ color |
|---|---|---|---|
| 1 | Silver | 3 | 0.3 |
| 2 | Black | 3 | 0.3 |
| 3 | Blue | 1 | 0.1 |
| 4 | Black | 3 | 0.3 |
| 5 | Gray | 2 | 0.2 |
| 6 | Silver | 3 | 0.3 |

In this dataset the best column for Count/Frequency -> color/supplier

Why?

- Color is nominal (no order)
- Only 5 categories → small number
- Count: Silver=3, Black=3, Blue=1, Gray=2,White=1
- Frequency: Silver=0.3, Black=0.3, Blue=0.1, Gray=0.2,White=0.1

# Comparison for nominal

| Feature | One-Hot Encoding (OHE) | Frequency Encoding |
|---|---|---|
| Cardinality | Best for Low Cardinality (<5 categories). | Best for High Cardinality (>10 categories). |
| Output Dimension | Increases dimensionality (1 column per category). | Maintains dimensionality (1 numerical column total). |
| Risk of Issue | Risk of High Dimensionality and Sparsity. | Risk of Collision (two categories get the same frequency). |
| Model Type Fit | Linear (LogReg, SVM), NN, and Distance-based. | Tree-based (Random Forest, XGBoost, CatBoost). |

# Binary encoding

- Useful when cardinality is high as one hot encoding will create a greater number of columns
- This saves vast amounts of memory and speeds up model training when dealing with high-cardinality data.
- Binary encoding doesn’t assume linear relationship between categories hence mitigating the risk of misleading linear models
- Minimum information loss: Compared to Frequency Encoding (which can merge information for categories with the same count), Binary Encoding preserves the uniqueness of every category without collision.
- Cons
  - Loss of Interpretability: The resulting binary features (columns) are abstract and do not directly relate to the original category.
  - Artificial Relationships: While it doesn't assume a simple linear order, the binary features still introduce a fixed numerical relationship between categories that may not exist. The model must learn that certain combinations of binary features correspond to distinct categories.
  - Not suitable for simple linear models

| ID | Product | Size | Color | Country | Category | Supplier | Price |
|---|---|---|---|---|---|---|---|
| 1 | Laptop | Small | Silver | USA | Premium | Dell | 1200 |
| 2 | Tablet | Medium | Black | China | Budget | Lenovo | 300 |
| 3 | Phone | Small | Blue | India | Premium | Samsung | 900 |
| 4 | Monitor | Large | Black | Japan | Standard | Sony | 400 |
| 5 | Laptop | Medium | Gray | Germany | Premium | HP | 1100 |
| 6 | Phone | Small | Silver | China | Budget | Xiaomi | 500 |
| 7 | Monitor | Large | White | USA | Standard | Dell | 450 |
| 8 | Tablet | Medium | Black | India | Budget | Samsung | 320 |
| 9 | Laptop | Large | Silver | Japan | Premium | Lenovo | 1250 |
| 10 | Phone | Small | Gray | Germany | Premium | Xiaomi | 850 |

| ID | Supplier | Supplier_ord | Sup_bin_0 | Sup_bin_1 | Sup_bin_2 |
|---|---|---|---|---|---|
| 1 | Dell | 1 | 0 | 0 | 1 |
| 2 | Lenovo | 2 | 0 | 1 | 0 |
| 3 | Samsung | 3 | 0 | 1 | 1 |
| 4 | Sony | 4 | 1 | 0 | 0 |
| 5 | HP | 5 | 1 | 0 | 1 |
| 6 | Xiaomi | 6 | 1 | 1 | 0 |

In this dataset the best column for effect encoding -> Supplier or country

Why?

- Supplier has 5 different companies (nominal)
- OHE would create 5 columns and binary 3.
- Works well for mid-level and high-level cardinality features
- Binary Encoding is good for Tree-based algorithms (Random Forest, Gradient Boosting) and Neural Networks that can learn the non-linear relationship between the new binary features and the target variable.

# Target encoding

- Target Encoding is for converting high cardinality nominal categorical features into numerical values.
- Target/ mean or likelihood encoding  leverages the relationship between the categorical feature and the target variable itself and replaces each category with the average value of the target variable observed for that category.
- Enc(i) = Mean(Target |Category = i)
  - If the target is binary (0 or 1), the encoded value is the proportion of observations in that category belonging to the positive class.
  - If the target is continuous (like price), the encoded value is the average value of the target for that category.
- Overfitting/Data Leakage: If a category appears only once or twice, the calculated mean perfectly reflects the target value of that few observations. In a predictive model, this leaks information from the target variable, making the model overly optimistic and generalizing poorly to new/unseen data.
- Sensitivity to Noise: A category appearing once is treated with the same confidence as a category appearing 1,000 times, leading to unstable predictions (high variance).

| ID | Product | Size | Color | Country | Category | Supplier | Price |
|---|---|---|---|---|---|---|---|
| 1 | Laptop | Small | Silver | USA | Premium | Dell | 1200 |
| 2 | Tablet | Medium | Black | China | Budget | Lenovo | 300 |
| 3 | Phone | Small | Blue | India | Premium | Samsung | 900 |
| 4 | Monitor | Large | Black | Japan | Standard | Sony | 400 |
| 5 | Laptop | Medium | Gray | Germany | Premium | HP | 1100 |
| 6 | Phone | Small | Silver | China | Budget | Xiaomi | 500 |
| 7 | Monitor | Large | White | USA | Standard | Dell | 450 |
| 8 | Tablet | Medium | Black | India | Budget | Samsung | 320 |
| 9 | Laptop | Large | Silver | Japan | Premium | Lenovo | 1250 |
| 10 | Phone | Small | Gray | Germany | Premium | Xiaomi | 850 |

In this dataset the best column for effect encoding -> Category (as it strongly influences price)

Why?

- Category influences price
- Target (Price) is numeric
- Very powerful in tree models (LightGBM, CatBoost, XGBoost)

- The values in the Category_TargetEnc column are the result of Target Encoding the Category column, using the unmodified Price column as the Target instead of a binary flag (like Price > 1000$).
- **Mean Target Encoding**, where the categorical feature is replaced by the average value of the target variable for all observations within that category.

| Category | IDs | Prices | Count (ni) | Sum of Prices (Si) | Category_TargetEnc (Si/ni) |
|---|---|---|---|---|---|
| **Premium** | 1, 3, 5, 9, 10 | 1200, 900, 1100, 1250, 850 | 5 | 5300 | 5300 / 5 = 1060 |
| **Budget** | 2, 6, 8 | 300, 500, 320 | 3 | 1120 | 1120 / 3 = 373.333 |
| **Standard** | 4, 7 | 400, 450 | 2 | 850 | 850 / 2 = 425 |

# Regularization Technique

**To mitigate the overfitting issue:**

- **Smoothing :** This blends the category's mean with the overall **global mean** of the target, using a smoothing factor (prior) to pull small-sample means closer to the global average. This stabilizes the encoding and handles rare categories
- **K-Fold Cross-Validation (Out-of-Fold Encoding):** This is the gold standard. The encoding for any data point is calculated using the mean from a different subset of the training data (the other K-1 folds). This prevents data leakage by ensuring an observation's target value never contributes to its own feature encoding.
- **Adding Random Noise:** A small amount of random Gaussian noise can be added to the encoded means to break the perfect relationship and reduce the tendency to overfit. This increases generalization as noise acts as a "distractor," forcing the model to learn the general trend rather than the exact mean value.

# Smoothing Target Encoding

- Smoothing (Bayesian mean) implements a weighted average that blends the category's specific mean with the overall Global Mean of the target variable.
- The weight assigned to the category's mean is based on its count (support).
- The Smoothed Formula

$$\text{Smoothed Mean} = \frac{\text{Count} \times \text{Category Mean} + \text{Prior} \times \text{Global Mean}}{\text{Count} + \text{Prior}}$$

  - *Smoothed mean*: The final smoothed encoded value for category (posterior mean)
  - *Category mean*($\mu_i$): The raw Category Mean (Target Mean for category *i*).
  - *Global mean*: The Global Mean (Target Mean for the entire dataset)-(prior belief).
  - Count: The number of records in the category (evidence)
  - Prior- Smoothing parameter (or regularization factor), often chosen through cross-validation or set based on domain knowledge. It acts as a **pseudo-count** that determines how much weight is given to the Global Mean.

**Effect of Smoothing**

**High Count (ni):** If a category has a large count (e.g., 'Premium' with ni=5), its Category Mean dominates the equation, and the smoothed mean stays close to original.

**Low Count (ni):** If a category has a small count (e.g., 'Standard' with ni=2), the Prior (m=3) has a stronger influence, pulling the smoothed mean closer to the Global Mean, thus stabilizing the encoding- less prone to overfitting

| Category | IDs | Prices | Count (ni) | Sum of Prices (Si) | Category_TargetEnc (Si/ni) |
|---|---|---|---|---|---|
| **Premium** | 1, 3, 5, 9, 10 | 1200, 900, 1100, 1250, 850 | 5 | 5300 | 5300 / 5 = 1060 |
| **Budget** | 2, 6, 8 | 300, 500, 320 | 3 | 1120 | 1120 / 3 = 373.333 |
| **Standard** | 4, 7 | 400, 450 | 2 | 850 | 850 / 2 = 425 |

| Category | Count (ni) | Category Mean (μi) | Calculation | Smoothed Encoded Value |
|---|---|---|---|---|
| **Premium** | 5 | 1060 | {(5 x1060.00) + (3 x 727.00)}/ (5 + 3) | **935.12** |
| **Budget** | 3 | 373.33 | {(3 x373.33) + (3 x 727.00)}/ (3 + 3) | **550.17** |
| **Standard** | 2 | 425 | {(2 x425.00) + (3 x 727.00)}/ (2 + 3) | **606.2** |

| Category | Original Mean (μi) | Smoothed Encoded Value | Pull Towards Global Mean (727.00) |
|---|---|---|---|
| **Premium** (ni=5) | **1060** | **935.12** | Only slightly pulled down (by $124.88) because of its high count. |
| **Standard** (ni=2) | **425** | **606.2** | Significantly pulled up (by $181.20) because its count is small (ni=2 is less than m=3). |
| **Budget** (ni=3) | **373.33** | **550.17** | Moderately pulled up (by $176.84) since n_i=m. |

Global Mean Price : The average price of all 10 products is 727.00.

Prior : We will use a smoothing parameter of 3.

# Hash encoding

- Feature Hashing, or the Hashing Trick, is a method used primarily for encoding nominal categories, especially those with very high cardinality (especially in NLP).
- It allows you to map large vocabulary into a fixed, manageable number of columns. This dramatically reduces the dimensionality while still retaining most of the distinguishing information.
- Like One-Hot encoding, the hash encoder converts the category into binary numbers using new data variables but here we can fix the number of new data variables (no dramatical increase in dimension).
- Hashing is highly memory efficient.
- Cons: Hash collision: Since a large number of unique words are mapped to a smaller number of feature bins, two different words (e.g., "apple" and "apricot") might end up being assigned to the exact same feature column. The model can no longer distinguish between these two words, which introduces a small amount of noise or ambiguity

| ID | Product | Size | Color | Country | Category | Supplier | Price |
|---|---|---|---|---|---|---|---|
| 1 | Laptop | Small | Silver | USA | Premium | Dell | 1200 |
| 2 | Tablet | Medium | Black | China | Budget | Lenovo | 300 |
| 3 | Phone | Small | Blue | India | Premium | Samsung | 900 |
| 4 | Monitor | Large | Black | Japan | Standard | Sony | 400 |
| 5 | Laptop | Medium | Gray | Germany | Premium | HP | 1100 |
| 6 | Phone | Small | Silver | China | Budget | Xiaomi | 500 |
| 7 | Monitor | Large | White | USA | Standard | Dell | 450 |
| 8 | Tablet | Medium | Black | India | Budget | Samsung | 320 |
| 9 | Laptop | Large | Silver | Japan | Premium | Lenovo | 1250 |
| 10 | Phone | Small | Gray | Germany | Premium | Xiaomi | 850 |

In this dataset the best column for effect encoding -> Supplier

Why?

- high cardinality (here 6 but if 500 OHE needs 500 columns )
- Here let's choose K=3 even though we have 6 suppliers
- Apply hypothetical hash function and calculate index= Hash {Supplier Name}) / mod(3)
- NLP or large category problems

Introduces hash collision: By reducing the dimensionality from 6 unique suppliers to just 3 feature columns, we forced distinct categories to share the same feature bin:

Collision 1 (Feature 0): Samsung and Xiaomi mapped to the same column.
Collision 2 (Feature 1): Dell and Sony mapped to the same column.
Collision 3 (Feature 2): Lenovo and HP mapped to the same column.

| ID | Supplier | Hypothetical Hash Value | Modulo (mod3) (Index) | Feature 0 | Feature 1 | Feature 2 |
|---|---|---|---|---|---|---|
| 1 | Dell | 1000 | 1000 mod(3)=1 | 0 | 1 | 0 |
| 2 | Lenovo | 1001 | 1001 mod(3)=2 | 0 | 0 | 1 |
| 3 | Samsung | 1002 | 1002 mod(3)=0 | 1 | 0 | 0 |
| 4 | Sony | 1003 | 1003 mod(3)=1 | 0 | 1 | 0 |
| 5 | HP | 1004 | 1004 mod(3)=2 | 0 | 0 | 1 |
| 6 | Xiaomi | 1005 | 1005 mod(3)=0 | 1 | 0 | 0 |

| ID | Product | Size | Color | Country | Category | Supplier | Price |
|---|---|---|---|---|---|---|---|
| 1 | Laptop | Small | Silver | USA | Premium | Dell | 1200 |
| 2 | Tablet | Medium | Black | China | Budget | Lenovo | 300 |
| 3 | Phone | Small | Blue | India | Premium | Samsung | 900 |
| 4 | Monitor | Large | Black | Japan | Standard | Sony | 400 |
| 5 | Laptop | Medium | Gray | Germany | Premium | HP | 1100 |
| 6 | Phone | Small | Silver | China | Budget | Xiaomi | 500 |
| 7 | Monitor | Large | White | USA | Standard | Dell | 450 |
| 8 | Tablet | Medium | Black | India | Budget | Samsung | 320 |
| 9 | Laptop | Large | Silver | Japan | Premium | Lenovo | 1250 |
| 10 | Phone | Small | Gray | Germany | Premium | Xiaomi | 850 |

| Encoding Type | Best Column | Reason |
|---|---|---|
| **Ordinal / Label** | Size | Natural order (Small < Medium < Large) |
| **One-Hot** | Color | No order, low cardinality |
| **Effect Encoding** | Product | Great for nominal mid-cardinality & regression interpretation |
| **Binary Encoding** | Supplier | Medium cardinality → reduces dimensions |
| **Hash Encoding** | Supplier | Show collision concept; scalable for larger vocabularies |
| **Target Encoding** | Category | Strongly correlated with Price → ideal use case |

Linear logic pulls categories toward the mean at a constant rate based on m

The category_encoders library uses this Sigmoid weight. Trusts the category mean almost completely once $n_i$ passes a certain threshold, but ignores it almost completely if the sample size is too small.

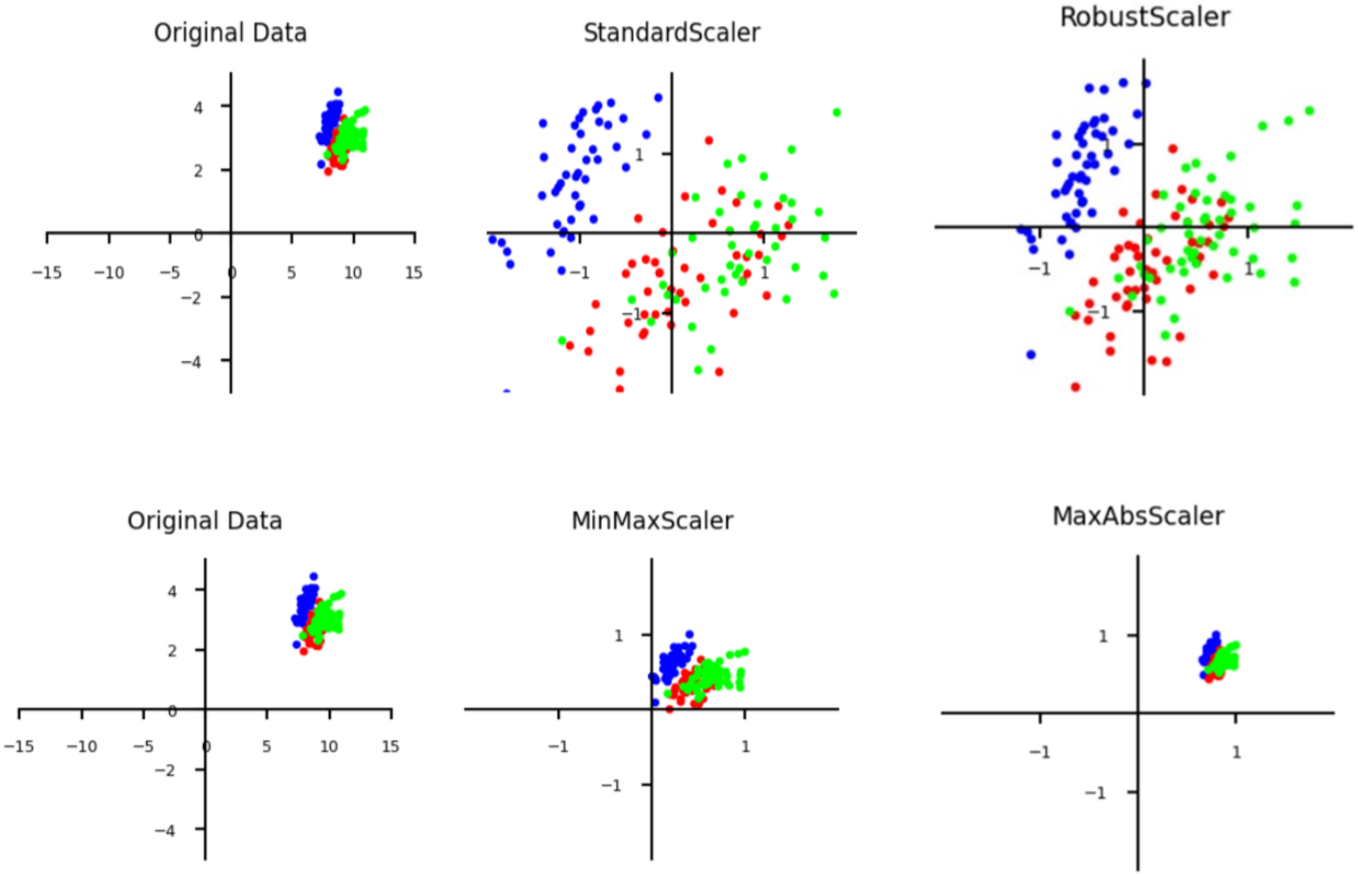
$$\zeta = \frac{1}{1 + \exp\left(-\frac{n_i - \text{min\_samples\_leaf}}{\text{smoothing}}\right)}$$

# Data/Feature scaling

- Changes the magnitude and range of the data but preserves the shape of the underlying distribution.
- Use when different numeric features have different scales (different range of values)
  - Features with much higher values may overpower the others
- Goal: bring them all within the same range
- Key point: Interpretability
- Mechanism: Applies a linear transformation (subtraction/division) to compress or expand the data based on its mean, standard deviation, min, or max.
- Why it's used: To ensure all features contribute equally to the distance calculation in distance-based algorithms (KNN, SVM) and to speed up the convergence of gradient-based optimizers.

# Scaling methods comparison

| Scaler | Best suited | Mechanism | Impact on Outliers | Preserves Sparsity (Zeros)? |
|---|---|---|---|---|
| **Standardization** (StandardScaler) | Gaussian (Normal) distributions | Removes mean, scales to unit variance | **Sensitive:** Outliers distort the mean and scale. | **No:** Shifts data to center at 0. |
| **Normalization** (MinMaxScaler) | Non-Gaussian data; known bounds (e.g., pixels) | Scales values to a fixed range (usually [0, 1]) | **Highly Sensitive:** One outlier squashes all other data. | **No:** Shifts data unless the min is already 0. |
| **Robust Scaling** (RobustScaler) | Data with many outliers or heavy skew | Uses Median and Interquartile Range (IQR) | **Resistant:** Outliers have little effect on the scaling. | **No:** Shifts data to center at the median. |
| **Max Absolute** (MaxAbsScaler) | Sparse data (mostly zeros) or text data | Divides by the absolute maximum value | **Sensitive:** High-magnitude outliers dominate the scale. | **Yes:** Zero remains zero. |

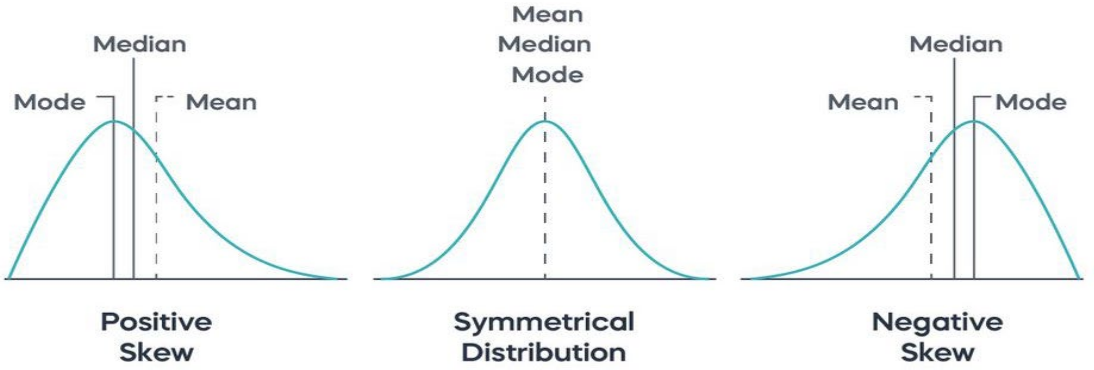

Original Data
StandardScaler
RobustScaler
Original Data
MinMaxScaler
MaxAbsScaler

# For property tax data

| Scaler | Mathematical Logic | Best Use Case | Behavior with Outliers | Resulting Range |
|---|---|---|---|---|
| **StandardScaler** | $z = \frac{x - \mu}{\sigma}$ | **Default choice.** Linear models, SVMs, and PCA. | **Sensitive.** Outliers skew the mean and shrink the "normal" data. | Typically -3 to 3 (but no hard limit). |
| **MinMaxScaler** | $x_{std} = \frac{x - x_{min}}{x_{max} - x_{min}}$ | **Neural Networks** and Image processing. | **Very Sensitive.** One outlier squishes all other data to near zero. | Exactly 0 to 1. |
| **RobustScaler** | $x_{scaled} = \frac{x - Q_2}{IQR}$ | **"Messy" data** with extreme outliers (e.g., your $50k tax house). | **Resistant.** Uses median/IQR, so outliers don't "pull" the scale. | Not fixed, but centers the bulk of data. |
| **MaxAbsScaler** | Based on max value | **Sparse Data** (Text/NLP). Keeps zeros as zeros. | **Sensitive.** The maximum value (outlier) dictates the scale. | -1 to 1. |

# Need of power transformation

- In machine learning, many algorithms assume that features follow a Gaussian (Normal) distribution (the "bell curve").
- However, real-world data is often "messy"—it might be skewed, have varying variance, or follow a power law.
- **Power Transformation** is a family of mathematical techniques used to map data from any distribution to as close to a Gaussian distribution as possible.
- Most parametric models (Linear Regression, Logistic Regression, LDA, Gaussian Naive Bayes) perform poorly when data is heavily skewed because:
  - Violation of Assumptions: These models assume that the "errors" or the features themselves are normally distributed.
  - Sensitivity to Magnitude: Skewed data often has a few very high values (long tails) that act like outliers, pulling the model's "logic" away from the majority of the data.
  - Heteroscedasticity: This is a fancy word meaning the "noise" in your data isn't constant. Power transforms help stabilize this variance so the model treats small and large values with equal importance



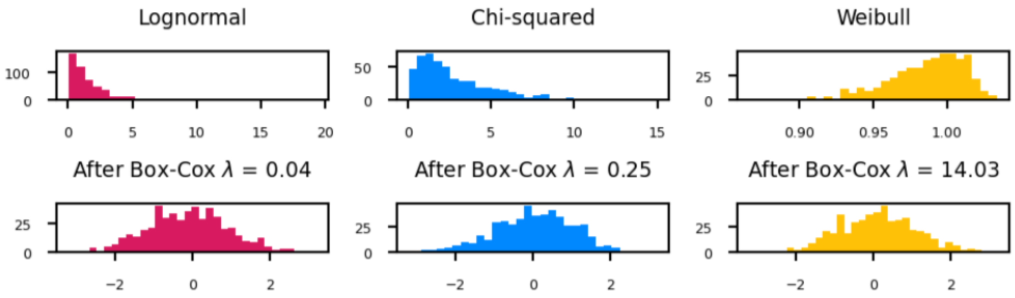

**Skewness** is a statistical measure that describes the **asymmetry** of a probability distribution around its mean. In simpler terms, it tells you if your data is "leaning" to one side or if it is perfectly balanced like a bell curve

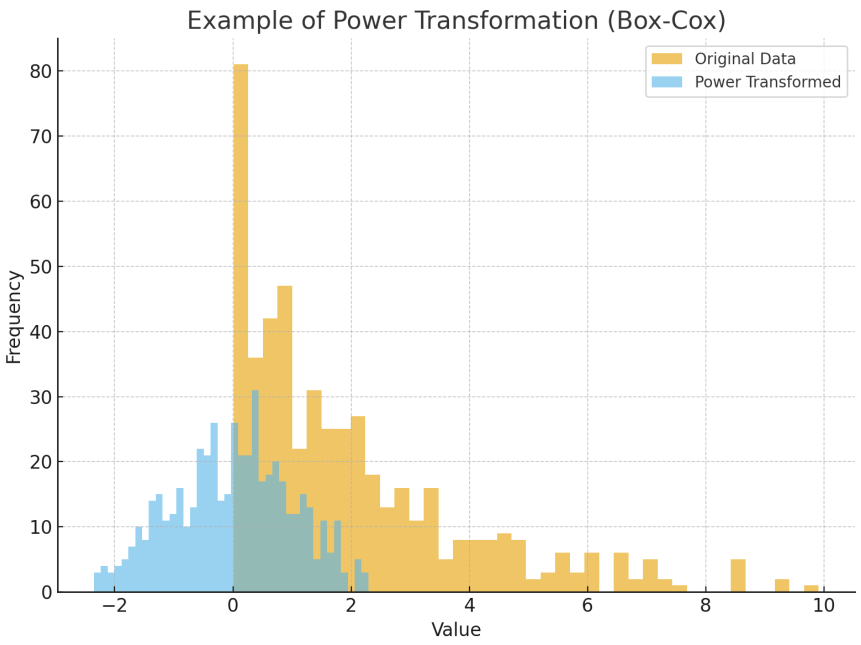
| Type | Visual Characteristic | How to fix | Examples |
|---|---|---|---|
| **Positive Skew** (Right-Skewed) | Long tail on the **right** side. Most data is bunched on the left. | fixed by taking roots or logs. | Household income, house prices, mileage on used cars. |
| **Negative Skew** (Left-Skewed) | Long tail on the **left** side. Most data is bunched on the right. | fixed by using powers (squares/cubes). | Age at retirement, exam scores (where most students pass). |
| **Zero Skew** (Symmetric) | Perfectly balanced "Bell Curve." | | Human heights, weights of standardized products. |

# Power transformation

- Some features follow certain distributions
  - E.g. number of twitter followers is log-normal distributed
- Box-Cox transformations transform these to normal distributions (λ is fitted)
  - Only works for positive values, use Yeo-Johnson otherwise (positive, zero, and negative values.)

$$bc_\lambda(x) = \begin{cases} log(x) & \lambda = 0 \\ \frac{x^\lambda - 1}{\lambda} & \lambda \neq 0 \end{cases}$$






Use it when your data shows:

- Strong skew (especially right-skew)
- Non-constant variance
- Heavy-tailed distributions
- Poor performance in linear models due to non-normality

Typical ML models that benefit:

- Linear Regression
- Logistic Regression
- KNN
- Distance-based models

**Original data (yellow histogram)**

- Very right-skewed
- Typical of exponential-type data (e.g., waiting times, time-to-failure)

**Power-transformed data (blue histogram)**

- More symmetric
- Skewness is reduced
- This improves model performance and reduces the effect of extreme values

[PowerTransformer — scikit-learn 1.7.2 documentation]

# Advanced Feature Engineering

### Feature Creation (Derived Features):

Example: Creating Age from Date of Birth or Price/SqFt from Price and Area.

### Textual Feature Extraction (NLP):

TF-IDF: Measures word importance based on frequency in a document vs. corpus.

Word2Vec: Creates dense vectors that capture semantic word relationships.

### Dimensionality Reduction:

PCA (Principal Component Analysis): Reduces complexity while retaining variance.

# Impact of Bias on Fairness

Bias directly influences whether the model treats different groups equitably.

- **Historical Bias:** Data reflects past inequalities. E**x:** Past hiring records favor men → hiring model predicts men as better candidates.
- **Sampling Bias:** Some groups are underrepresented. **Ex:** Face recognition dataset mostly contains light-skinned faces → model performs poorly on darker-skinned individuals.
- **Label Bias:** The labels themselves are unfair or reflect human judgment errors. Ex:  Loan default labels based on biased approval systems.
- **Measurement Bias:** Features are measured differently across groups. **Ex:** Different medical devices work better on one demographic than another.

# Consequences on fairness

**Unequal Error Rates:** Model may misclassify or reject certain groups disproportionately. Ex: A credit scoring model misclassifies 20% of applications from group A but only 5% from group B.
**Discrimination in Automated Decisions:** Biased models/data can reinforce inequality in: Hiring, Loan approvals, Insurance pricing, Policing algorithms
**Loss of Trust & Ethical Issues:** Users lose confidence when systems produce unfair outcomes.
**Legal & Compliance Risks:** Regulations (e.g., GDPR, fairness guidelines) require equal treatment across protected groups.

# Impact of Bias on Model Generalization

**Overfitting to Dominant Groups**

When data overrepresents one group, the model *specializes* in it and performs poorly elsewhere.

Example: A medical diagnosis model trained mainly on adults → fails on children.

**Poor Feature Learning**

Biased data may push the model to learn shortcuts.

Example: A model predicts "wolf vs dog" based on background snow instead of the animal → fails on new images.

**Domain Shift Problems**

If training data is not diverse, the model cannot adapt to new environments.

Example: Self-driving car trained only in sunny conditions → fails in rain or night-time.

**Unrealistic Assumptions (Algorithmic Bias)**

Simplified linear assumptions may generalize poorly when reality is more complex.

Example: Using linear models for non-linear relationships.

# How to Reduce Bias, Improve Fairness, Generalization

- Improve dataset diversity
- Balanced sampling and reweighting
- Fair labeling practices
- Use fairness-aware algorithms
- Cross-group evaluation metrics (FPR, FNR, demographic parity)
- Regular auditing and bias detection tools
- Human-in-the-loop review for high-risk decisions

# Bias mitigation techniques

| Description | Why It's Used | Pros & Cons |
|---|---|---|
| **Reweighing (or Re-sampling):** Assigning a higher weight to samples from an underrepresented (unprivileged) group that received an unfavorable outcome, and a lower weight to overrepresented samples. | Used to achieve statistical parity or demographic parity, ensuring that the model doesn't learn correlations between a protected attribute (like gender or race) and the target outcome simply due to unequal group representation. | **Pros:** Algorithm-agnostic (works with any model); conceptually simple to implement.<br>**Cons:** Does not alter the features, so proxy bias (where the model uses non-protected features correlated with the protected attribute) can still exist; may lead to overfitting on the weighted/re-sampled minority group. |

- **Pre-processing Techniques** (Intervene on the Data):These methods modify the training data before it's fed into the model to reduce or eliminate bias.
- **Reweighing (or Re-sampling)**

# Oversampling

- Oversampling is a method used to balance the dataset when the protected group (unprivileged group) or the unfavorable outcome (e.g., loan rejection) is significantly underrepresented.
- **Identify the Minority:** Determine the group or outcome that is underrepresented (e.g., female applicants, or all rejected applicants regardless of gender).
- **Duplicate/Synthesize:** New samples are created or the existing samples belonging to the minority class are **duplicated** until the class distribution is more balanced, often matching the size of the majority class.
- **Advantage:** oversampling is used to enforce **Statistical Parity** or **Demographic Parity** by ensuring that the model is trained on an equal or fairer representation of all groups, preventing the model's performance from being unfairly skewed toward the majority or privileged group.

| Description | Why It's Used | Pros & Cons |
|---|---|---|
| **SMOTE (Synthetic Minority Over-sampling Technique):** SMOTE generates a new synthetic data point by taking two similar minority points, calculating the distance between them, and creating a new point **along the line segment** connecting the two neighbors. | Used to **increase the representation** of a minority group/outcome and help the model better define the decision boundary for that group without simply repeating the same data, which can lead to overfitting. | **Pros:** Algorithm-agnostic; prevents the model from ignoring the minority class; **synthetic data** helps generalize better than simple duplication. **Cons:** Can create "noise" if the minority samples are already close to the majority boundary; the new synthetic data points may not accurately reflect the real-world distribution. |

| Technique | Description | Why It's Better Than SMOTE | Pros & Cons |
|---|---|---|---|
| **ADASYN (Adaptive Synthetic Sampling)** | Generates more synthetic points for hard-to-learn (majority-surrounded) minority samples and fewer for easy ones. It adaptively shifts the decision boundary toward the majority class. | **Adaptive Focus:** Directs sampling effort to the critical decision boundary where points are likely misclassified, shifting the boundary more effectively. | **Pros:** Better boundary accuracy. **Cons:** Sensitive to noise/outliers; risks increasing class overlap. |
| **SMOTE-ENN (SMOTE with Edited Nearest Neighbors)** | **Hybrid method:** 1. Oversamples (SMOTE). 2. Removes noisy or overlapping samples (ENN) from *both* classes. | **Boundary Cleanup:** Cleans up noisy synthetic points and majority intrusions, creating sharper, better-defined class boundaries. | **Pros:** Clearer boundaries; high accuracy. **Cons: Removes original data** (majority points); more complex to implement. |

# Conclusion & Key Takeaways

- Feature Engineering is both an Art & Science requiring domain knowledge and experimentation.
- Better Features → Better Models: Effective transformation leads to simpler, more powerful models.
- Actionable Advice: Choose scaling/encoding methods based on algorithm and data distribution.