

COMS4995W32

Applied Machine Learning

Dr. Spencer W. Luo

Columbia University | Fall 2025



Data

Agenda



- Motivation
- Data Cleaning
- Data Transformation
- Feature Engineering
- Feature Selection
- Case Study



Motivation


Why Data Matters 🤔



- Garbage in → Garbage out
- ML models are only as good as their data
- 80% of project time = data prep (industry stats)

Model Quality = Data Quality



- Clean + consistent data → reliable predictions
- Poor + noisy quality data → misleading results
-  Even the BEST algorithms **fail** with messy data

Real-World Dirty Data



- Missing values → blanks, “N/A”
- Noisy values → typos, sensor errors
- Inconsistent formats → units, codes, duplicates

id	age	income	state	rating
1	25	80k	MARS	5
2	0	?	US-NY	N/A
3	-99	70,000	NY	4



Data Cleaning

Why Clean Data? 🧹



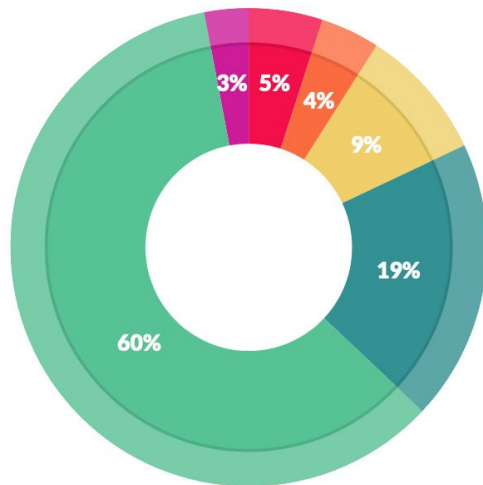
- Dirty data → wrong insights
 - Inconsistent records can mislead the model, producing unreliable predictions
- 80% of ML effort = data preparation
 - Kaggle studies show the majority of project time is spent collecting, cleaning, and transforming data - not training fancy models
- Clean data = reliable models
 - Well-prepared datasets reduce bias, improve generalization

Why Clean Data? 🧹



How a Data Scientist Spends Their Day

Here's where the popular view of data scientists diverges pretty significantly from reality. Generally, we think of data scientists building algorithms, exploring data, and doing predictive analysis. That's actually not what they spend most of their time doing, however.



What data scientists spend the most time doing

- Building training sets: 3%
- Cleaning and organizing data: 60%
- Collecting data sets; 19%
- Mining data for patterns: 9%
- Refining algorithms: 4%
- Other: 5%


Types of Dirty Data



- **Missing** → empty values, “N/A”
 - Caused by skipped survey questions, sensor downtime, or data entry omissions
 - Leads to incomplete patterns and biased results
- **Noisy** → typos, random errors
 - Human mistakes (e.g., “25O00” instead of “25000”), faulty sensors
 - Adds variance and hides real signal
- **Inconsistent** → units, duplicates
 - Same info in different formats (lbs vs kg, “NY” vs “New York”)
 - Duplicate or conflicting records distort analysis

Missing Values - Simple Fixes



- Drop rows/columns (if sparse)
- Fill with constants (e.g., “Unknown”)
- Mean / Median / Mode imputation
-  Imputation must be done on **TRAINING DATA ONLY**

Missing Values - Smarter Fixes



- Class-wise mean/median
- Predictive models (regression, kNN)
- Add a “missing_flag” column

Noisy Data - Cause 🤔



- Faulty sensors
 - Hardware malfunction
 - Calibration drift
- Data entry mistakes
 - Copy-paste errors, wrong decimal placement
- Transmission errors
 - 👉 Example: temperature = -273 °C 😄


Handling Noisy Data



- Binning → smooth by mean/median
 - `data = bin(data).mean()`
- Regression smoothing
 - `data = fit_regression(x, y).predict(x)`
- Clustering → detect/remove anomalies
 - `anomalies = cluster(data) == outlier`

Inconsistent Data



- Example: USD vs RMB, lbs vs kg
- Schema mismatches (same field, diff names)
- Deduplication required
 -  Visual idea: picture of “kg → lbs” conversion

Summary: Data Cleaning ✓



- Missing, noise, inconsistency = universal problems
- Strategies: drop, impute, smooth, normalize
- Domain knowledge + reproducibility are critical



Data Transformation

Why Transform Data?



- Raw data often not ready for models
- Transformation improves comparability & stability
- Visualization helps spot issues early

Normalization



- Scale values into $[0,1]$ range
 - Rescales each feature so the smallest becomes 0 and the largest becomes 1
- Useful for distance-based methods (kNN, clustering)
 - Ensures all features contribute equally to distance calculations; prevents large-scale features from dominating
- Sensitive to outliers
 - Extreme values stretch the range, making most data points compressed into a narrow band



Standardization



- Transform data to mean = 0, std = 1
 - Centers each feature around zero and rescales by its standard deviation
- Works better for many ML models
 - Especially useful for linear regression, logistic regression, SVMs, and neural networks that assume standardized inputs
- Less sensitive to outliers vs normalization
 - Outliers still affect the mean/variance, but the effect is weaker than min–max scaling

Comparison



Aspect	Normalization 	Standardization 
What it does	Scale values into [0,1] range	Shift to mean = 0 and scale to std = 1
Effect	Compresses data into a fixed interval	Centers and balances features
Good for	Distance-based methods (kNN, clustering)	Linear/Logistic Regression, SVMs, Neural Nets
Outliers	Very sensitive (extremes dominate scaling)	Less sensitive, still influenced
Intuition	“Stretch/squash” to fit between 0 - 1	“Re-center & rescale” for comparability


Visualization for Cleaning



- Histograms → spot missing values & skewed distributions
- Scatterplots → reveal outliers & correlations
- Boxplots → highlight anomalies within groups

Summary: Transform



- Transform: normalize, standardize etc.
- Visualize: histograms, scatterplots, boxplots
- Goal: make data reliable, interpretable, ready for ML
 -  Flow: Raw Data → Transform → Visualization → **Signals**



Feature Engineering

What is a Feature? 🔍



- Representation of raw data → input to algorithms
- Converts context into numbers
- Example:
 - 👉 “age” → bucket “young/middle/old”
 - 👉 [Visual] raw → feature transformation diagram



Why Feature Engineering?



- Expose structure hidden in raw data
 - Turn messy signals into meaningful patterns
 - e.g., timestamps → day of week, hour
- Improve model performance & interpretability
 - Well-designed features reduce noise, boost accuracy, and make results easier to explain
- Often more impactful than algorithm choice
 - A simple model with strong features can **outperform** a complex model with poor inputs



Categorical Features



- One-hot encoding → binary vectors
 - Create a new column for each category, with 1 if the row belongs to that category and 0 otherwise
 -  Preserves all categories without assuming order
 -  Can lead to high-dimensional, sparse data if categories are many

Categorical Features



- Target encoding → replace with mean target
 - Replace each category with the average value of the target variable for that category
 -  Compact representation, powerful for tree-based models
 -  Risk of leakage if target statistics are computed using the full dataset — must be done on training folds only

Encoding Comparison



Method	Example (color)	Pros 👍	Cons ⚠️
One-hot	Red \rightarrow [1,0,0]	Simple no leakage	High dimensional Sparse
Target encoding	Red \rightarrow 0.72	Compact powerful for trees	Risk of leakage if not careful

Text Features



- Tokenization basics: words, subwords
 - Break text into smaller units (e.g., “cats” → [“cat”, “##s”]); choice affects vocabulary size and coverage
- Bag-of-Words → high-dimensional & sparse
 - Represent documents by raw word counts
 - Easy to implement but ignores order and meaning

Text Features 🖋️



- TF-IDF → weigh distinctive words
 - Scale word importance by frequency in document vs across corpus; highlights informative terms (e.g., “fraud” in reviews)
- Embeddings → semantic meaning (Word2Vec, BERT)
 - Map words into dense vectors where similar meanings cluster together; capture context, semantics, and relations

Image Features



- Pixels as raw input (matrix)
 - Treat an image as a 2D (grayscale) or 3D (RGB) numeric tensor
 - models consume normalized pixel intensities

Image Features



- Simple: color histograms, edges
 - Global stats capture tone/distribution
 - Edge detectors (e.g., Sobel) highlight structure/contours
- Advanced: CNN embeddings
 - Pre-trained CNNs map images to compact, semantic vectors



Video Features



- Frames as sequences
- Temporal differences \rightarrow motion features

Industry Insights




- Simple, domain-informed features often beat fancy models
- “Better data and features trump fancier algorithms — again and again.”
- Key takeaway:
 - Do NOT underestimate the power of simple and interpretable features

Summary: Feature Engineering



- Features = bridge from data → ML
- Good features matter more than complex models
- Avoid leakage & redundancy

 Flow: Raw Data → Features → Model



Feature Selection

Why Select Features? ✂️



- Reduce overfitting
- Improve training speed
- Increase interpretability

Common Methods



- Decision trees → built-in feature importance
 - Tree models (Random Forest, Gradient Boosted Trees) rank features by how much they reduce impurity
 - Naturally highlight the most informative splits
 - Importance scores can be biased toward high-cardinality features, so interpret with care

Commo Methods



- LASSO regularization → zero-out weak features
 - Adds an L1 penalty term to the loss function
 - Forces coefficients of less useful features exactly to zero, performing selection during training
 - Works well for high-dimensional, sparse data



Case Study