SRI SATHYA SAI INSTITUTE OF HIGHER LEARNING
(Deemed to be University)
Department of Mathematics and Computer Science

CONSOLIDATED LECTURE NOTES - 1

Unit 1-10 of Textbook 1

UDSC-402: Data Analysis and Visualization	Instructor: Dr. D Bhanu Prakash	Date: 01-08-2026

Chapter 1 — Data Preparation in a Machine Learning Project

The Applied Machine Learning Process (4 Steps)

Every predictive modeling project — regardless of the specific dataset — follows the same general four-step process:

- Step 1: Define Problem — frame the task (classification, regression, or another problem type), collect the relevant data, and understand it through summary statistics and visualization.
- Step 2: Prepare Data — transform the raw, collected data into a form usable for modeling.
- Step 3: Evaluate Models — design a robust test harness: choose a performance metric, establish a baseline, choose a resampling technique (train-test split or k-fold cross-validation), and perform hyperparameter tuning / ensembling.
- Step 4: Finalize Model — select the final model (model selection), evaluate on a hold-out validation set, summarize performance for stakeholders, and productionize the model.

Key Definition

Data Preparation: The transformation of raw data into a form that is more suitable for modeling. Also referred to as data wrangling, data munging, data cleaning, data pre-processing, or (loosely) feature engineering.

Why Raw Data Cannot Be Used Directly

- Machine learning algorithms require input data to be numbers.
- Some algorithms impose specific requirements on the data (e.g., a particular probability distribution, no correlated inputs).
- Statistical noise and errors in the data may need to be corrected.
- Complex nonlinear relationships in the data may need to be exposed/teased out.

The Five Core Data Preparation Tasks (detailed fully in Chapter 3)

- Data Cleaning — identifying and correcting mistakes or errors in the data.
- Feature Selection — identifying the input variables most relevant to the prediction task.

- Data Transforms — changing the scale or distribution of variables.
- Feature Engineering — deriving new input variables from the available data.
- Dimensionality Reduction — creating compact, lower-dimensional projections of the data.

Guiding philosophy: data preparation should best expose the unknown underlying structure of the prediction problem to the learning algorithm — this is why it is treated as a step of discovery, not a fixed recipe.

Chapter 2 — Why Data Preparation Is So Important

Data (in an ML context): Rows of observations and columns of variables (structured/tabular data), where a machine learning algorithm learns a mapping from the input variables to a target (output) variable.

Three Reasons Data Preparation Matters

- Machine Learning Algorithms Expect Numbers — text, categories, and dates must be encoded numerically before modeling.
- Machine Learning Algorithms Have Requirements — e.g., a Gaussian-distributed input, the absence of missing values, or uncorrelated input features.
- Model Performance Depends on Data — the chosen representation of data (scaling, engineered features) can influence performance as much as, or more than, the choice of algorithm itself.

Key Takeaway

Predictive modeling is, in practice, mostly data preparation. Practitioners typically spend the majority of a project's time and effort on this step because it directly determines how well the underlying structure of a problem is exposed to the learning algorithm.

Chapter 3 — Tour of Data Preparation Techniques (Core Taxonomy)

This chapter is the backbone of the subject — memorize this taxonomy carefully.

3.1 Data Cleaning

Data Cleaning: Fixing systematic problems or errors in messy data, typically guided by domain expertise.

- Using statistics to define "normal" data and identify outliers (Ch. 6).
- Identifying & removing columns with the same value / zero variance, and duplicate rows (Ch. 5).
- Marking empty values as missing (Ch. 7).
- Imputing missing values using statistics or a learned model (Ch. 8, 9, 10).

Data cleaning is typically performed first, before other data preparation steps.

3.2 Feature Selection

Feature Selection: Techniques for selecting a subset of input features that are most relevant to the target variable, to avoid misleading the model with irrelevant/redundant inputs and to favor simpler models.

Taxonomy of feature selection methods:

- Unsupervised — does not use the target variable (e.g., removing highly correlated inputs).
- Supervised — uses the target variable, and is further divided into:
  - Intrinsic — feature selection happens automatically as part of model fitting (e.g., LASSO regression, decision trees).
  - Wrapper — explicitly searches for the subset of features that produces the best-performing model (e.g., Recursive Feature Elimination, RFE).
  - Filter — scores each input feature independently (typically with a statistical measure such as correlation) and selects the highest-scoring subset.

3.3 Data Transforms

Data Transform: A technique that changes the type or the probability distribution of a data variable.

Data type taxonomy:

- Numeric Data Type: Integer (no fractional part), Float (floating point values).
- Categorical Data Type: Ordinal (ranked labels), Nominal (unranked labels), Boolean (True/False).

Transform	Definition
Discretization Transform	Encode a numeric variable as an ordinal variable.
Ordinal Transform	Encode a categorical variable into an integer variable.
One Hot Transform	Encode a categorical variable into binary (dummy) variables.
Normalization Transform	Scale a variable to the range 0 to 1.
Standardization Transform	Scale a variable to a standard Gaussian (mean = 0, std = 1).
Power Transform	Change the distribution of a variable to be more Gaussian.
Quantile Transform	Impose a probability distribution (uniform or Gaussian) on a variable.

Important: transforms are generally fit and applied per variable, and the fitted transform object must be saved (along with the final model) to correctly process new/future data.

3.4 Feature Engineering

Feature Engineering: The process of creating new input variables from the available data, typically requiring subject-matter expertise.

- Adding a boolean flag variable for some state.
- Adding a group or global summary statistic (e.g., a mean).

- Adding new variables for each component of a compound variable (e.g., splitting a date-time into year/month/day).
- Polynomial Transform — creating copies of numerical input variables raised to a power (or multiplied together).

3.5 Dimensionality Reduction

Dimensionality: The number of input features for a dataset; more input variables define a higher-dimensional feature space.

Curse of Dimensionality: As the number of dimensions (features) increases, the data becomes an increasingly sparse and unrepresentative sample of that space.

Dimensionality Reduction: Creating a projection of the data into a lower-dimensional space that preserves the most important properties of the original data; unlike feature selection, projected variables are not directly interpretable in terms of the original inputs.

Matrix-factorization based methods:

- Principal Component Analysis (PCA)
- Singular Value Decomposition (SVD)

Model-based methods:

- Linear Discriminant Analysis (LDA)
- Autoencoders

Manifold learning methods:

- Self-Organizing Maps (SOM)
- t-Distributed Stochastic Neighbor Embedding (t-SNE)

The main effect of matrix-factorization methods is removing linear dependencies (correlations) between input variables.

Chapter 4 — Data Preparation Without Data Leakage

Data Leakage: A problem where information about the holdout (test/validation) dataset is made available to the model during training, giving it an unrealistic advantage and producing an overly optimistic or otherwise incorrect estimate of model performance.

The Naive (INCORRECT) Approach

- 1. Prepare / transform the entire dataset (e.g., normalize using the global min and max of all rows).
- 2. Split the data into train and test sets.
- 3. Evaluate the model.

This causes indirect data leakage: summary statistics (min, max, mean, standard deviation) computed over the whole dataset leak information about the test set into the training process — even though the test rows are never directly used to fit the model.

The Correct Approach — Train/Test Split

- 1. Split the data into train and test sets FIRST.
- 2. Fit the data-preparation transform (e.g., a scaler) on the TRAINING set only.
- 3. Apply the fitted transform to both the training and test sets.
- 4. Evaluate the model.

The Correct Approach — k-Fold Cross-Validation

Data preparation must be fit within each fold, using only that fold's training portion, then applied to that fold's held-out portion. In practice this is achieved with a Pipeline (an ordered sequence of transform steps ending in a model), which is passed as a single unit to the cross-validation function (e.g., cross_val_score()).

Key principle: "The entire modeling pipeline must be prepared only on the training dataset" — so this kind of evaluation is really pipeline evaluation, not merely model evaluation.

k-Fold Cross-Validation — Recap

Split the dataset into k non-overlapping folds. Train the model on k−1 folds and test it on the remaining (held-out) fold. Repeat so every fold is used as the test set exactly once, and report the average performance across all folds.

Repeated Stratified k-Fold: Cross-validation repeated multiple times (for robustness), where each fold preserves the overall class ratio of the dataset (stratified) — considered best practice for classification problems.

k-fold cross-validation generally gives a more reliable performance estimate than a single train-test split, at the cost of higher computation (repeated model fitting).

Chapter 5 — Basic Data Cleaning

Always run these basic checks first on any new dataset.

1. Columns That Contain a Single Value

Zero-Variance Predictor: A column where every row has the exact same value; it carries no information for modeling and can also cause errors in some algorithms.

Detection: use the number of unique values per column (e.g., Pandas nunique()); a count of 1 flags the column for removal.

2. Columns With Very Few Unique Values

Near-Zero-Variance Predictor: A column whose variance is not exactly zero but is very small — e.g., only 2 to 9 unique numeric values spread across thousands of rows.

These are not automatically useless — they may represent legitimate categorical/ordinal information. Detection: compute the percentage of unique values per column = (number of unique values ÷ total rows) × 100, and flag columns below a threshold (commonly < 1%).

3. Columns With Low Variance

The statistical variance (average squared deviation from the mean) of a column can be used directly as a filter — e.g., scikit-learn's VarianceThreshold removes columns whose variance falls below a chosen cut-off.

4. Duplicate Rows

Identical rows add no new information and can bias a model or waste computation. Identify with duplicated() and remove with drop_duplicates().

Exam tip: Chapter 5 targets uninformative columns/rows (constant, near-constant, duplicate) — this is distinct from handling missing values (Ch. 7–10) and outliers (Ch. 6).

Chapter 6 — Outlier Identification and Removal

Outlier: An observation that lies an abnormal distance from other values in a dataset; an unlikely event under the assumed data-generating distribution. Causes include measurement variability, experimental error, natural rare events, and data-entry mistakes.

(a) Standard Deviation Method — for Gaussian / near-Gaussian data

Empirical rule (68–95–99.7 rule) for a Gaussian distribution:

- 1 standard deviation from the mean → covers ≈ 68% of the data.
- 2 standard deviations from the mean → covers ≈ 95% of the data.
- 3 standard deviations from the mean → covers ≈ 99.7% of the data.

z-score: The number of standard deviations a value x_i is from the mean μ.

$$\mathbf{z = (x_i - \mu) / \sigma}$$

Common rule of thumb: values with $|z| > 3$ (outside mean ± 3σ) are flagged as outliers. Use 2σ for smaller samples and 4σ for larger samples.

cut_off = std × 3lower = mean − cut_offupper = mean + cut_offoutlier if value < lower OR value > upper

(b) Interquartile Range (IQR) Method — for non-Gaussian data

Percentiles/quartiles: sort the data; the 25th percentile (Q1), 50th percentile (median), and 75th percentile (Q3) divide it into four groups.

IQR (Interquartile Range): IQR = Q3 − Q1 — represents the middle 50% (the "body") of the data; forms the box in a box-and-whisker plot.

lower = Q1 − k × IQRupper = Q3 + k × IQR

- Common factor k = 1.5 → ordinary outliers.
- Factor k = 3 (or more) → "extreme outliers" / "far outs".

(c) Automatic (Model-Based) Outlier Detection

Framed as one-class classification: the model characterizes what "normal" data looks like and flags deviations.

Local Outlier Factor (LOF): Scores each point by how isolated it is relative to the local density of its k-nearest neighbors; a larger score indicates a more likely outlier. Works well for low/moderate-dimensional feature spaces but degrades under the curse of dimensionality. In scikit-learn, LocalOutlierFactor.fit_predict() returns +1 for inliers and −1 for outliers.

Other model-based detectors: Isolation Forest, One-Class SVM, Elliptic Envelope.

Multivariate Extension

For multiple Gaussian variables, the outlier boundary generalizes to an ellipse/ellipsoid (standard-deviation method) or a hyper-rectangle (IQR method) in multi-dimensional space.

Practical workflow: fit the outlier detector on the training set only (to avoid data leakage — same principle as Chapter 4), remove/mask flagged rows, then fit and evaluate the model. This commonly improves error metrics such as Mean Absolute Error (MAE).

Chapter 7 — How to Mark and Remove Missing Data

Missing Data: Rows where one or more column values are absent, sometimes represented with a placeholder/sentinel character (e.g., "?", empty string, 0, "NULL") instead of being genuinely blank. Causes: unrecorded observations, data corruption, equipment malfunction, or merging incompatible datasets.

Step 1 — Mark Missing Values

Replace sentinel/placeholder characters with a proper missing-value marker (NaN) — e.g., DataFrame.replace('?', nan) — so that pandas, NumPy, and scikit-learn correctly recognize them as missing (checked with isnull()/isna()).

Why Missing Values Are a Problem

Most machine learning algorithms cannot operate on data containing NaN, and will raise an error when the model is fit — so missing values must always be handled (removed or imputed) before modeling.

Step 2 — Remove Rows With Missing Values (Listwise Deletion)

The simplest strategy: drop any row containing one or more NaN values, using dropna().

Downsides: can discard a large fraction of the dataset (and potentially valuable information); only viable if enough complete rows remain; not applicable when missingness is spread across most rows.

This limitation motivates the more sophisticated Imputation techniques covered in Chapters 8–10 (filling in estimated values instead of discarding data).

Chapter 8 — How to Use Statistical Imputation

Data Imputation (Missing Data Imputation): Replacing missing values in a dataset with substituted, estimated values so that the complete dataset can be used with algorithms requiring no missing data.

Statistical Imputation: Calculating a statistic per column (from the present, observed values in that column) and replacing all missing values in that column with the computed statistic.

Common Statistics Used

- Column mean
- Column median
- Column mode (most frequent value)
- A constant value

Scikit-Learn Implementation — SimpleImputer

The strategy parameter selects: 'mean', 'median', 'most_frequent', or 'constant'.

Leakage-safe workflow: fit the imputer's statistics on the TRAINING data only, then transform both train and test data (same principle as Chapter 4) — typically wrapped inside a Pipeline together with the model for correct cross-validation.

Different strategies (mean/median/mode/constant) should be compared empirically for a given dataset, since the best statistic to use is data-dependent.

Chapter 9 — How to Use KNN Imputation

KNN (Nearest Neighbor) Imputation: A model-based imputation approach where, for each missing value in a feature, a k-Nearest Neighbors model finds the k most similar training rows (based on a distance measure computed over the other features) and averages their values for that feature to estimate the missing value.

More sophisticated than statistical imputation because it exploits relationships between features rather than relying on a single global column statistic.

Configuration / Hyperparameters

- Distance measure (commonly Euclidean distance).

- Number of neighbors, k.

Scikit-Learn Implementation — KNNImputer

Used similarly to SimpleImputer (fit / transform); must be fit only on the training set to avoid data leakage, and is best wrapped inside a Pipeline. The choice of k should be tuned/compared — a larger k is not always better for downstream model performance.

Chapter 10 — How to Use Iterative Imputation

Iterative Imputation: Each feature with missing values is modeled as a function of all other features (a regression problem), and missing values are predicted feature-by-feature (sequentially), reusing previously imputed values as inputs for imputing subsequent features. The entire process is repeated (iterated) multiple times so that estimates for all features are progressively refined.

Also Known As

- Fully Conditional Specification (FCS)
- Multivariate Imputation by Chained Equations (MICE)

Key Configuration Details

- A regression algorithm (often a simple linear model) is fit per feature to predict its missing values from the other features.
- Number of iterations: typically kept small (commonly ≈10, sometimes 10–20) — a tunable hyperparameter.
- Imputation order (the sequence in which features are processed) is configurable — e.g., ascending order of missing-value count — and can affect the result.

Scikit-Learn Implementation — IterativeImputer

Located in sklearn.impute (requires enabling via sklearn.experimental.enable_iterative_imputer). Used with fit/transform, fit on training data only, and typically embedded in a Pipeline for correct cross-validation.

Comparing the Three Imputation Approaches (Chapters 8–10)

Method	Basis	Chapter
Statistical Imputation	Single column statistic: mean / median / mode / constant	8
KNN Imputation	Average of the k nearest-neighbor rows	9
Iterative Imputation (MICE)	Regression model per feature, iterated across all features	10

Quick-Revision Summary — All 10 Chapters

Ch	Topic	Must-Know Definition / Algorithm
1	ML Process & Data Prep	4-step process: Define → Prepare → Evaluate → Finalize. Data preparation = transforming raw data into a form suitable for modeling.
2	Importance of Data Prep	Data must be numeric and meet algorithm requirements; model performance depends heavily on data representation.
3	Taxonomy	5 tasks: Cleaning; Feature Selection (filter / wrapper / intrinsic); Transforms (normalize, standardize, power, quantile, one-hot, ordinal, discretize); Feature Engineering (polynomial); Dimensionality Reduction (PCA, SVD, LDA).
4	Data Leakage	Fit every data-prep step on training data only; use a Pipeline inside cross-validation.
5	Basic Cleaning	Remove zero-variance columns; examine near-zero-variance columns; remove duplicate rows.
6	Outliers	Std-Dev method (z-score, 3σ rule); IQR method (Q1 − 1.5×IQR, Q3 + 1.5×IQR); LOF / Isolation Forest for automatic detection.
7	Missing Data	Mark sentinel characters as NaN; listwise deletion (dropna) is the simplest fix.
8	Statistical Imputation	SimpleImputer — mean / median / mode / constant.
9	KNN Imputation	KNNImputer — average of k nearest neighbors.
10	Iterative Imputation	IterativeImputer / MICE — per-feature regression, repeated iterations.

Golden Rule (Chapters 4–10): Every data preparation step (scaling, imputing, outlier removal, feature selection) must be fit on training data only, then applied to test data — never fit on the full dataset before splitting, or you introduce data leakage.