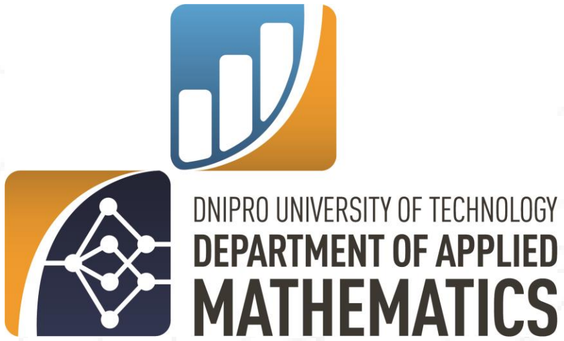


Data Preparation and ML Pipeline

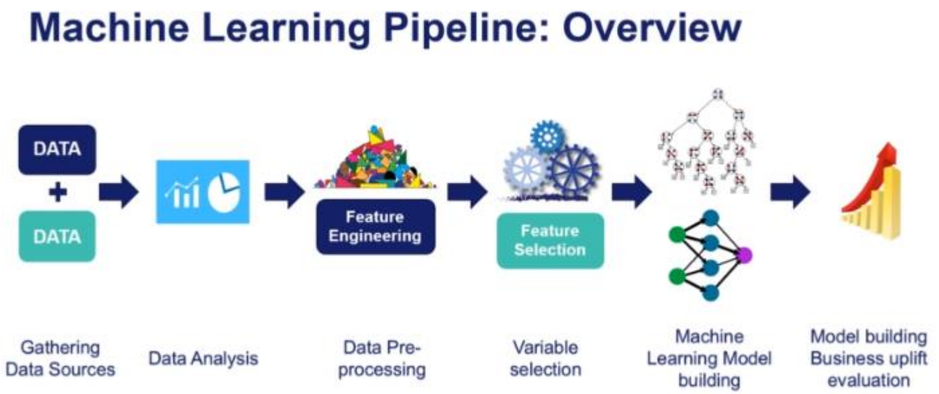
Course "Machine Learning: From Mathematical Foundations to Implementation in PYTHON"

Lecture by prof. Dmytro Babets

1. Learning Objectives

After this lecture, you will be able to:

1. Explain the purpose and main stages of data preparation.
2. Formulate mathematical descriptions of normalization, standardization, and encoding.
3. Identify sources of missing data and anomalies, and apply mathematical tools for their detection.
4. Describe the structure of a machine learning (ML) pipeline and explain its role in reproducibility and validation.
5. Implement preprocessing and model integration in Python using `scikit-learn`.



Guangyuan's Research and Development Blog: Overview of ML Pipelines

Data preparation forms the **foundation of the ML workflow**. Regardless of the algorithm's sophistication, the quality of predictions depends heavily on the **consistency, scale, and structure** of the input data.

In this lecture, we will:

- review **mathematical principles** of preprocessing operations,
- learn to handle **missing values and anomalies**,
- and understand how preprocessing can be **automated** in a structured **ML pipeline**.

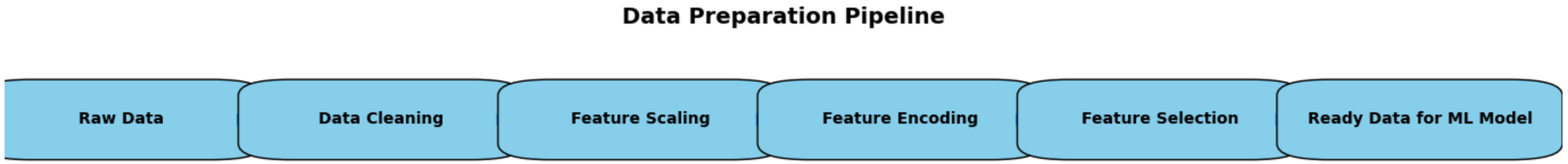
Mathematical rigor in these steps is crucial because normalization, scaling, and encoding are **transformations of the feature space**, which directly influence the geometry of the data and, consequently, the optimization landscape of ML algorithms.

Theoretical Background of Data Preparation

Goal: Transform raw data $X = \{x_i\}_{i=1}^{n} \subset \mathbb{R}^m$ into a consistent numerical representation suitable for ML models.

Main Stages:

1. Data Cleaning
2. Feature Scaling
3. Feature Encoding
4. Feature Selection



Data preparation is the process of mapping

$$T: X_{\text{raw}} \to X_{\text{ready}},$$

where T is a sequence of transformations ensuring:

- consistency of measurement scales,
- robustness to missing data,
- and numerical suitability for optimization-based learning methods (e.g., gradient descent).

Step 1 — Data Cleaning

Cleaning tasks:

- Detect and handle missing values.
- Remove duplicates and inconsistencies.
- Identify implausible or impossible values.

Mathematical Representation:

Let

$$M_{ij} = \begin{cases} 1, & \text{if } x_{ij} \text{ is missing,} \\ 0, & \text{otherwise.} \end{cases}$$

Imputation replaces x_{ij} when $M_{ij} = 1$ by a function:

$$x_{ij}^{*} = f_j(X_{\text{observed}}).$$

The **mask matrix** $M \in \{0,1\}^{n \times m}$ identifies missing entries.

Typical imputation functions f_j include:

- **Mean imputation:** $f_j = \bar{x}_j$.
- **Median imputation:** $f_j = \text{median}(x_j)$.
- **Model-based imputation:** regression or k-NN imputation:

$$x_{ij}^{*} = \frac{\sum_{k \in N_i} w_{ik} x_{kj}}{\sum_{k \in N_i} w_{ik}},$$

where N_i — nearest neighbors of sample i, w_{ik} — similarity weights.

These substitutions are based on the **assumption of statistical continuity** of features and minimal distortion of the original data distribution.

Step 2 — Feature Scaling

Main techniques:

1. Normalization:

$$x' = \frac{x - x_{\min}}{x_{\max} - x_{\min}} \in [0, 1]$$

2. Standardization:

$$x' = \frac{x - \mu}{\sigma}$$

3. Robust scaling (resistant to outliers)

$$x' = \frac{x - \text{median}(x)}{\text{IQR}(x)}$$

where IQR is the interquartile range.

Scaling transforms the **geometry** of the feature space.

Consider a dataset $X \subset \mathbb{R}^m$. Many ML algorithms (SVMs, k-NN, gradient-based models) rely on **distance measures** such as Euclidean norm:

$$d(x_i, x_j) = \| x_i - x_j \|_2.$$

Without scaling, features with large magnitude dominate this distance metric, biasing the learning process.

- **Normalization** preserves the shape but rescales to a fixed range.
- **Standardization** centers data around zero with unit variance, making each feature equally influential.
- **Robust scaling (resistant to outliers)**.

,

Step 3 — Feature Encoding

Why Encoding?

- ML algorithms require **numeric input**.
- Categorical features must be **transformed** into numbers while preserving information.
- Improper encoding may distort **distances**, **correlations**, and **model interpretability**.

Example:

$$X_{raw} = [\text{"red"}, \text{"green"}, \text{"blue"}] \Rightarrow X_{encoded} = [?, ?, ?]$$

Feature encoding is the process of **mapping categorical variables** (nominal or ordinal) into a **numerical feature space** that a machine learning algorithm can process.

Let's denote a categorical feature: $x_j \in \{c_1, c_2, \ldots, c_K\}$, where *K* is the number of unique categories.

Encoding defines a function: $E_j : \{c_1, \ldots, c_K\} \to \mathbb{R}^d$, that embeds symbolic categories into a **vector space** of dimension d, suitable for further processing by algorithms relying on distance or gradient-based optimization.

Two main types of encoding are used:

- **Ordinal encoding** (for ordered categories),
- **One-hot or binary encoding** (for unordered categories).

Ordinal Encoding

Ordinal feature: Categories have a *natural order*.

Example: Education level

"high *school*" < *"bachelor"* < *"master"* < *"PhD"*

Encoding: $$E(x_j) = \begin{cases} 1, & \text{if } x_j = \text{“high school”} \\ 2, & \text{if } x_j = \text{“bachelor”} \\ 3, & \text{if } x_j = \text{“master”} \\ 4, & \text{if } x_j = \text{“PhD”} \end{cases}$$

Category	Meaning	Encoded Value
very bad	lowest satisfaction	1
bad		2
neutral		3
good		4
excellent	highest satisfaction	5

$$E(x_j) = \begin{cases} 1, & \text{if } x_j = \text{“very bad”} \\ 2, & \text{if } x_j = \text{“bad”} \\ 3, & \text{if } x_j = \text{“neutral”} \\ 4, & \text{if } x_j = \text{“good”} \\ 5, & \text{if } x_j = \text{“excellent”} \end{cases}$$

For **ordinal features**, the categories possess an **inherent ranking**, so the encoding preserves the **monotonic relationship**:

$$x_{i,j_1} < x_{i,j_2} \Rightarrow E(x_{i,j_1}) < E(x_{i,j_2}).$$

Mathematically, the ordinal encoder defines a **monotone mapping** $E: C \to \mathbb{R}$, ensuring that distances between encoded values respect the order, though not necessarily the magnitude of differences.

Caution: ordinal encoding **introduces artificial distances** - the difference between encoded values 1 and 2 may not represent the same semantic gap as between 3 and 4.

Thus, it should be used **only for truly ordered** categorical features.

Key Point

Ordinal encoding assumes **ordered categories** where numerical differences have *rank meaning* but not *metric meaning* - i.e.,

$E(x_j = "medium") - E(x_j = "low")$ indicates a higher level, but not necessarily *twice as much* in value.

One-Hot Encoding

Nominal feature: No natural order between categories.

Example 1: color ∈ {red, green, blue}

One-hot encoding:

$$E(color) = [1 \quad 0 \quad 0],\ [0 \quad 1 \quad 0],\ [0 \quad 0 \quad 1]$$

Example 2: country ∈ {*Poland, Germany, France, Italy*}

country	[Poland]	[Germany]	[France]	[Italy]
Poland	1	0	0	0
Germany	0	1	0	0
France	0	0	1	0
Italy	0	0	0	1

$$E(\text{country}) = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

For **nominal (unordered)** features, the encoder expands each category into a **binary vector**:

$$E: \{c_1, \dots, c_K\} \to \{0,1\}^K,$$

such that the vector has a 1 in the position of the corresponding category and 0 elsewhere.

This mapping is equivalent to creating **K basis vectors** of the canonical Euclidean space $\mathbb{R}^K$:

$$E(c_k) = e_k = [0, \dots, 1, \dots, 0]^T.$$

Advantages:

- No artificial order introduced.
- Distances between different categories are **equal** (orthogonal vectors).

Drawbacks:

- Dimensionality explosion when K is large.
- Sparse representation (many zeros), which may affect computational efficiency.

To mitigate this, techniques like **hash encoding** or **embedding representations** can be used.

Binary Encoding

Binary Encoding:

•Convert category index into binary representation.

•Compact form of one-hot encoding.

Example 1. City Names

city $\in \{London, Paris, Rome, Berlin, Madrid, Warsaw, Vienna, Prague\}$

There are $K = 8$ categories $\rightarrow$ use $\lceil \log_2 8 \rceil = 3$ bits.

City	Integer Code	Binary Encoding
London	1	[0,0,1]
Paris	2	[0,1,0]
Rome	3	[0,1,1]
Berlin	4	[1,0,0]
Madrid	5	[1,0,1]
Warsaw	6	[1,1,0]
Vienna	7	[1,1,1]
Prague	8	[0,0,0]

Let c_k correspond to integer k. Represent it in binary form of length $\lceil \log_2 K \rceil$:

$$E(c_k) = \text{binary}(k)$$

For 5 categories,

$$E(c_1) = [0,0,1], E(c_2) = [0,1,0], E(c_5) = [1,0,1].$$

This reduces dimensionality while maintaining uniqueness.

Example 2. Blood Type

$$\text{blood} \in \{A, B, AB, O\}$$
$$K = 4 \Rightarrow \lceil \log_2 4 \rceil = 2$$

Blood Type	Integer	Binary
A	1	[0,1]
B	2	[1,0]
AB	3	[1,1]
O	4	[0,0]

Hash Encoding

Hash Encoding:

Map categories into a fixed number of dimensions via hash function:

$$E(c_k) = h(c_k) \bmod d$$

Example 1. **Movie Genres**

genre $\in \{Action, Comedy, Drama, Horror, Romance, Sci-Fi, Documentary\}$

Step 1. Choose the hash space size: $d = 4$.

$$h(genre) = (sum\ of\ character\ codes) \bmod 4$$

Step 2. Compute a hash index:

- Convert each letter to its ASCII (or Unicode) code.
- Sum all codes.
- Take the remainder when divided by 4.

Character	Code
A	65
c	99
t	116
i	105
o	111
n	110

Sum = 65 + 99 + 116 + 105 + 111 + 110 = **606**

Then:

$h("Action") = 606 \bmod 4 = 2$

Define a **hash function** $h: C \to \{0,1,\dots,d-1\}$.

Collisions may occur, but this approach enables efficient vectorization of large categorical spaces (e.g., text features).

Hashing is widely used in large-scale applications such as text vectorization (“hashing trick”) and online learning models.

Genre	Hash Index (h(genre))	Encoded Vector
Action	2	[0,0,1,0]
Comedy	1	[0,1,0,0]
Drama	2	[0,0,1,0] (collision with Action)
Horror	3	[0,0,0,1]
Romance	0	[1,0,0,0]
Sci-Fi	1	[0,1,0,0] (collision with Comedy)
Documentary	3	[0,0,0,1]

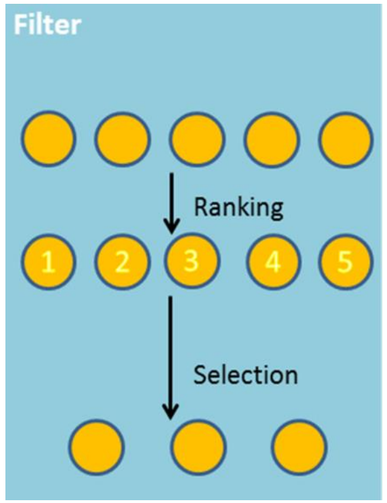
Feature Selection

Why Feature Selection and Dimensionality Reduction?

- High-dimensional data → increased model complexity
- Redundant or irrelevant features → overfitting
- Dimensionality reduction → better generalization and interpretability

Goal:

Simplify *model* + *Preserve information content*



Given a dataset

$$X \in \mathbb{R}^{n \times p},$$

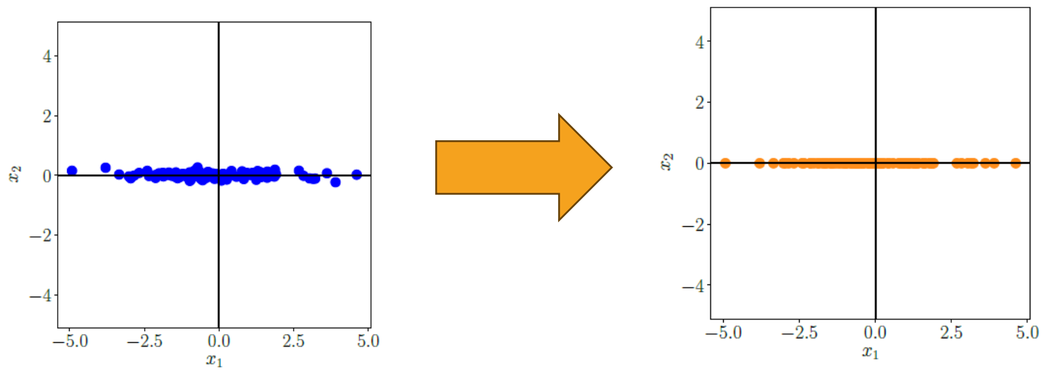
where n — number of samples, p — number of features.

When $p \gg n$ (high-dimensional regime), many features may be irrelevant or correlated.

Feature selection and dimensionality reduction aim to **reduce the effective dimension** $p' < p$ while preserving the most **informative subspace** of the data distribution.

Two main approaches:

1. **Feature Selection:** choose a subset of the original features.
2. **Dimensionality Reduction:** transform features into a new, lower-dimensional basis.



Filter methods — independent of model

Evaluate each feature's **relevance** using a score function:

$$S(x_j) = f(x_j, y),$$

such as:

- Pearson correlation coefficient:

$$r_j = \frac{\mathrm{cov}(x_j, y)}{\sigma_{x_j}\sigma_y}$$

- Mutual information:

$$I(x_j; y) = \sum_{x_j, y} p(x_j, y) \log \frac{p(x_j, y)}{p(x_j)p(y)},$$

where *p(x,y)* represents **the probability that both events happen simultaneously**

Keep top-k features according to $S(x_j)$.

Example

Suppose we have a small dataset predicting whether a student **passes (y=1)** or **fails (y=0)** an exam based on three features:

Student	Hours_Study (x_1)	Sleep_Hours (x_2)	Coffee_Cups (x_3)	Passed (y)
A	1	8	0	0
B	2	7	1	0
C	3	7	1	1
D	4	6	2	1
E	5	5	0	1

Feature Relevance via Pearson Correlation - **Top feature:** Hours_Study (strongest correlation)

Feature	Correlation with y	Interpretation
Hours_Study (x_1)	**+0.86**	Strong positive correlation → more study → higher chance to pass
Sleep_Hours (x_2)	**-0.72**	Strong negative correlation → less sleep → higher chance to pass
Coffee_Cups (x_3)	**+0.32**	Poor positive correlation → more coffee → higher chance to pass

Filter methods - Example

Feature Relevance via Mutual Information (MI)

Mutual Information measures how much **information** about y is provided by x_j, capturing **nonlinear** relationships.

$$I(x_j; y) = \sum_{x_j, y} p(x_j, y) \log \frac{p(x_j, y)}{p(x_j)p(y)}$$

compares the joint distribution $p(x, y)$ with what the joint *would be* if the two variables were independent $(p(x)p(y))$

- If $p(x, y) = p(x)p(y)$: they're **independent** $\rightarrow I(X; Y) = 0$
- If $p(x, y)$ differs a lot: they're **dependent** $\rightarrow I(X; Y) > 0$

Feature	Mutual Information (bits)	Interpretation
Hours_Study (x_1)	**0.28**	Most informative about "Pass/Fail"
Coffee_Cups (x_3)	0.11	Moderately informative
Sleep_Hours (x_2)	0.0	Non informative

Feature	Pearson	MI	Average Rank
Hours_Study	1	1	🥇
Sleep_Hours	2	3	🥈
Coffee_Cups	3	2	🥈

Method	Measures	Detects nonlinear?	Example Use
Pearson correlation	Linear association	❌	Regression, simple filters
Mutual information	Shared information	✅	Complex/nonlinear tasks
Both combined	Robust feature selection	✅	Recommended in pipelines

Dimensionality Reduction (PCA)

Goal:

Find *projection* $Z = XW, \quad W \in \mathbb{R}^{p \times k},\ k < p$

such that Z retains maximal information about X

Dimensionality reduction can be formulated as finding a mapping:

$$f: \mathbb{R}^p \to \mathbb{R}^k,$$

that minimizes **information loss** or preserves **variance**, **distances**, or **probability structure**.

PCA seeks orthogonal directions (principal components) that **maximize variance**.

Given centered data matrix X_c, compute covariance:

$$S = \frac{1}{n} X_c^\top X_c.$$

Then solve eigenvalue problem:

$$S w_j = \lambda_j w_j.$$

- w_j: principal directions
- λ_j: explained variance

The projection:

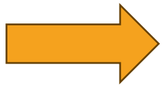
$$Z = X_c W_k,$$

where W_k contains top k eigenvectors.

Feature Reduction Impact:

Benefits:

- Improved generalization
- Reduced training time
- Simplified model interpretation



Mathematically:

Find $f(X): \mathbb{R}^p \to \mathbb{R}^k,\ k < p$

such that:

$I(Y; f(X)) \approx I(Y; X)$

Retain maximal information about target Y.

Step 1. Data Cleaning — Python Implementation Overview

Category	Class / Function	Library / Module	Main Parameters	Description / Purpose
Missing values detection	DataFrame.isnull() / DataFrame.isna()	pandas	—	Returns a Boolean mask showing which entries are missing (NaN).
	DataFrame.notnull() / DataFrame.notna()	pandas	—	Returns Boolean mask for non-missing entries.
	DataFrame.info()	pandas	—	Displays non-null counts and data types of each column — useful for quick missing-value inspection.
Counting missing values	DataFrame.isnull().sum()	pandas	—	Computes the total number of missing entries per column.
Handling missing values	DataFrame.dropna()	pandas	axis={0,1}, how={'any','all'}, subset, inplace	Removes rows or columns containing missing values.
	DataFrame.fillna()	pandas	value, method={'ffill','bfill'}, axis, inplace	Fills missing values with a specified constant or by forward/backward propagation.
Statistical imputation	SimpleImputer	sklearn.impute	missing_values, strategy={'mean','median','most_frequent','constant'}, fill_value	Replaces missing values with a computed statistic or a constant.
Advanced imputation	KNNImputer	sklearn.impute	n_neighbors, weights={'uniform','distance'}, metric	Imputes missing values using the k-nearest neighbors algorithm.
Iterative model-based imputation	IterativeImputer	sklearn.impute	estimator, max_iter, random_state	Predicts missing values by iteratively modeling each feature as a function of others.
Duplicate detection	DataFrame.duplicated()	pandas	subset, keep={'first','last',False}	Returns Boolean Series marking duplicated rows.
Duplicate removal	DataFrame.drop_duplicates()	pandas	subset, keep, inplace	Removes duplicate rows from the dataset.
Detecting implausible values	DataFrame.describe()	pandas	include, percentiles	Provides summary statistics for detecting out-of-range or unrealistic values.
	DataFrame.clip()	pandas	lower, upper	Limits values within specified bounds to remove extreme outliers.
Data type correction	DataFrame.astype()	pandas	dtype, errors={'raise','ignore'}	Converts columns to correct data types (e.g., categorical, numeric).
Replace inconsistent labels	DataFrame.replace()	pandas	to_replace, value, regex	Replaces incorrect or inconsistent categorical labels.

Feature Scaling — Python Implementation Overview

Category	Class / Function	Library / Module	Main Parameters	Description / Purpose
Standardization (Z-score scaling)	StandardScaler	sklearn.preprocessing	with_mean=True, with_std=True, copy=True	Scales features to zero mean and unit variance. Common for linear models, PCA, SVMs.
Min-Max normalization	MinMaxScaler	sklearn.preprocessing	feature_range=(0,1), clip=False	Rescales data to a fixed interval [a,b] Useful for neural networks.
Robust scaling (outlier-resistant)	RobustScaler	sklearn.preprocessing	with_centering=True, with_scaling=True, quantile_range=(25.0,75.0)	Scales using median and IQR (interquartile range). Reduces the influence of outliers.
Normalization (L1, L2, Max)	Normalizer	sklearn.preprocessing	norm={'l1','l2','max'}	Scales samples (rows) to have unit norm. Often used in text and distance-based models.
Quantile transformation (nonlinear scaling)	QuantileTransformer	sklearn.preprocessing	n_quantiles, output_distribution={'uniform', 'normal'}, random_state	Maps data to a uniform or Gaussian distribution using empirical quantiles. Reduces effect of outliers and skewness.
Power transformation (variance stabilization)	PowerTransformer	sklearn.preprocessing	method={'yeo-johnson','box-cox'}, standardize=True	Applies a power-law transform to make data more Gaussian-like. Useful for skewed positive data.
Column-wise application	ColumnTransformer	sklearn.compose	transformers, remainder	Allows applying different scalers to different feature subsets (e.g., numeric vs categorical).

Feature scaling ensures that all numerical features contribute **equally** to model learning and prevents those with large magnitudes from dominating gradient descent updates or distance computations.

Feature Encoding - sklearn

Class / Function	Description	Key Parameters	Example Usage
sklearn.preprocessing.LabelEncoder	Encodes categorical labels (strings) into integers (0, 1, 2, …). Used for target variables or single categorical columns.	_	le = LabelEncoder(); y = le.fit_transform(y_raw)
sklearn.preprocessing.OrdinalEncoder	Converts categorical features into integer values according to specified or learned ordering. Useful for ordinal features (e.g., education level).	categories='auto', dtype=float, handle_unknown='use_encoded_value'	enc = OrdinalEncoder(); X_enc = enc.fit_transform(X[['Education']])
sklearn.preprocessing.OneHotEncoder	Converts categorical features into one-hot (dummy) binary columns. Avoids ordinal relationships.	sparse_output=False, drop='first', handle_unknown='ignore'	ohe = OneHotEncoder(); X_ohe = ohe.fit_transform(X[['City']])
sklearn.compose.ColumnTransformer	Combines multiple encoders/scalers applied to different columns in a single pipeline.	transformers=[('ohe', OneHotEncoder(), cat_features), ('scaler', StandardScaler(), num_features)]	ct = ColumnTransformer([...]); X_proc = ct.fit_transform(X)

- LabelEncoder should only be used for target variables, not for input features (because it imposes false ordinality).
- OrdinalEncoder is best suited for ordered categories (e.g., “low < medium < high”).
- OneHotEncoder or pd.get_dummies are preferred for nominal categories.
- ColumnTransformer is essential for combining encoding and scaling in a single ML pipeline.

Feature Selection (Filter Methods) & Dimensionality Reduction (PCA)

Class / Function	Description	Key Parameters	Example Usage
sklearn.feature_selection.VarianceThreshold	Removes all features whose variance does not meet a specified threshold (useful for eliminating constant or quasi-constant features).	threshold=0.0	sel = VarianceThreshold(threshold=0.01); X_sel = sel.fit_transform(X)
sklearn.feature_selection.SelectKBest	Selects the top k features according to a statistical score (ANOVA F-value, chi-squared, mutual information, etc.).	score_func=f_classif, k=10	sel = SelectKBest(score_func=f_classif, k=5); X_new = sel.fit_transform(X, y)
sklearn.feature_selection.SelectPercentile	Selects a user-specified top percentage of features with the highest scores.	score_func=chi2, percentile=20	sel = SelectPercentile(score_func=chi2, percentile=20); X_new = sel.fit_transform(X, y)
sklearn.feature_selection.mutual_info_classif / mutual_info_regression	Computes the mutual information (information gain) between each feature and the target variable. Non-parametric and captures nonlinear relationships.	discrete_features='auto', n_neighbors=3, random_state=None	mi = mutual_info_classif(X, y); pd.Series(mi, index=X.columns)
sklearn.decomposition.PCA	Performs **Principal Component Analysis** to reduce dimensionality while preserving maximal variance. Linear orthogonal transformation.	n_components, svd_solver, whiten, random_state	pca = PCA(n_components=2); X_pca = pca.fit_transform(X_scaled)

- Filter methods evaluate each feature independently of others — fast and interpretable but may ignore feature interactions.
- Mutual Information is statistical dependency measurer used before model training.
- PCA finds new orthogonal components maximizing variance — mathematically based on eigenvalue decomposition of the covariance matrix.
- Choose the number of components k based on the explained variance ratio (pca.explained_variance_ratio_).

Titanic ML pipeline code (ML pipeline step by step)

```python
import pandas as pd
from sklearn.model_selection import train_test_split, cross_val_score
from sklearn.preprocessing import StandardScaler, OneHotEncoder
from sklearn.impute import SimpleImputer
from sklearn.compose import ColumnTransformer
from sklearn.pipeline import Pipeline
from sklearn.linear_model import LogisticRegression
from sklearn.datasets import fetch_openml
from sklearn.feature_selection import VarianceThreshold

# === 1. Load Titanic dataset ===
X, y = fetch_openml("titanic", version=1, as_frame=True, return_X_y=True)
y = y.astype(int)

print("\n--- INITIAL DATA (10 random samples) ---")
print(X.sample(10, random_state=42))
```

```
--- INITIAL DATA (10 random samples) ---
      pclass                                               name     sex   age  \
1148       3                            Rintamaki, Mr. Matti    male  35.0
1049       3                                Nakid, Mr. Sahid    male  20.0
982        3                           Lyntakoff, Mr. Stanko    male   NaN
808        3                                Ford, Mr. Arthur    male   NaN
1195       3                          Shaughnessy, Mr. Patrick    male   NaN
240        1   Romaine, Mr. Charles Hallace ('Mr C Rolmane')    male  45.0
1118       3                  Peltomaki, Mr. Nikolai Johannes    male  25.0
596        2                           Wilhelms, Mr. Charles    male  31.0
924        3                                Kelly, Mr. James    male  34.5
65         1  Chambers, Mrs. Norman Campbell (Bertha Griggs)  female  33.0

      sibsp  parch              ticket     fare cabin embarked boat  body  \
1148      0      0  STON/O 2. 3101273   7.1250   NaN        S  NaN   NaN
1049      1      1                2653  15.7417   NaN        C    C   NaN
982       0      0              349235   7.8958   NaN        S  NaN   NaN
808       0      0           A/5 1478   8.0500   NaN        S  NaN   NaN
1195      0      0              370374   7.7500   NaN        Q  NaN   NaN
240       0      0              111428  26.5500   NaN        S    9   NaN
1118      0      0  STON/O 2. 3101291   7.9250   NaN        S  NaN   NaN
596       0      0              244270  13.0000   NaN        S    9   NaN
924       0      0              330911   7.8292   NaN        Q  NaN  70.0
65        1      0              113806  53.1000    E8        S    5   NaN

                         home.dest
1148                           NaN
1049                           NaN
982                            NaN
808   Bridgwater, Somerset, England
1195                           NaN
240                   New York, NY
1118                           NaN
596                 London, England
924                            NaN
65        New York, NY / Ithaca, NY
```

Column	Type	Description
pclass	Categorical (1, 2, 3)	Passenger class (1st, 2nd, 3rd) — proxy for social/economic status.
sex	Categorical (male, female)	Passenger's gender.
age	Numerical	Passenger's age (may contain missing values).
sibsp	Numerical	Number of siblings/spouses aboard.
parch	Numerical	Number of parents/children aboard.
fare	Numerical	Ticket fare.
embarked	Categorical	Port of embarkation (C, Q, S).
survived	Target (binary)	1 = survived, 0 = died.

Titanic ML pipeline code (ML pipeline step by step)

```python
# Select all relevant features (drop text fields and identifiers)
X = X[['pclass', 'sex', 'age', 'sibsp', 'parch', 'fare', 'embarked']]

# Separate numeric and categorical columns
num_features = ['age', 'sibsp', 'parch', 'fare']
cat_features = ['pclass', 'sex', 'embarked']

# === 2. Data Cleaning ===
num_cleaner = SimpleImputer(strategy='median')
cat_cleaner = SimpleImputer(strategy='most_frequent')

X_num_cleaned = pd.DataFrame(num_cleaner.fit_transform(X[num_features]), columns=num_features)
X_cat_cleaned = pd.DataFrame(cat_cleaner.fit_transform(X[cat_features]), columns=cat_features)
X_cleaned = pd.concat([X_num_cleaned, X_cat_cleaned], axis=1)

print("\n--- AFTER DATA CLEANING (missing values replaced) ---")
print(X_cleaned.sample(10, random_state=42))

# === 3. Feature Scaling ===
scaler = StandardScaler()
X_scaled = X_cleaned.copy()
X_scaled[num_features] = scaler.fit_transform(X_scaled[num_features])

print("\n--- AFTER FEATURE SCALING (numeric standardized) ---")
print(X_scaled.sample(10, random_state=42))
```

```
--- AFTER DATA CLEANING (missing values replaced) ---
       age  sibsp  parch     fare pclass     sex embarked
1148  35.0    0.0    0.0   7.1250      3    male        S
1049  20.0    1.0    1.0  15.7417      3    male        C
982   28.0    0.0    0.0   7.8958      3    male        S
808   28.0    0.0    0.0   8.0500      3    male        S
1195  28.0    0.0    0.0   7.7500      3    male        Q
240   45.0    0.0    0.0  26.5500      1    male        S
1118  25.0    0.0    0.0   7.9250      3    male        S
596   31.0    0.0    0.0  13.0000      2    male        S
924   34.5    0.0    0.0   7.8292      3    male        Q
65    33.0    1.0    0.0  53.1000      1  female        S

--- AFTER FEATURE SCALING (numeric standardized) ---
            age     sibsp     parch      fare pclass     sex embarked
1148   0.426099 -0.479087 -0.445000 -0.505708      3    male        S
1049  -0.736663  0.481288  0.710763 -0.339111      3    male        C
982   -0.116523 -0.479087 -0.445000 -0.490805      3    male        S
808   -0.116523 -0.479087 -0.445000 -0.487824      3    male        S
1195  -0.116523 -0.479087 -0.445000 -0.493624      3    male        Q
240    1.201274 -0.479087 -0.445000 -0.130140      1    male        S
1118  -0.349075 -0.479087 -0.445000 -0.490240      3    male        S
596    0.116029 -0.479087 -0.445000 -0.392119      2    male        S
924    0.387341 -0.479087 -0.445000 -0.492093      3    male        Q
65     0.271064  0.481288 -0.445000  0.383183      1  female        S
```

Stage	Purpose	Mathematical Aspect	Transformation Example
Data Cleaning	Replace missing values using statistical measures	$x_i = \text{median}(X)$ or most frequent	NaN → 29.7
Feature Scaling	Normalize variance & remove unit dependency	$z = \frac{x-\mu}{\sigma}$	Age=22 → -0.43

Titanic ML pipeline code (ML pipeline step by step)

```python
# === 4. Feature Encoding ===
encoder = OneHotEncoder(handle_unknown='ignore')
encoded = encoder.fit_transform(X_scaled[cat_features])
encoded_df = pd.DataFrame(encoded.toarray(), columns=encoder.get_feature_names_out(cat_features))

X_encoded = pd.concat([X_scaled[num_features], encoded_df], axis=1)
print("\n--- AFTER FEATURE ENCODING (categorical → numeric) ---")
print(X_encoded.sample(10, random_state=42))

# === 5. Feature Selection (variance filter) ===
selector = VarianceThreshold(threshold=0.22)
X_selected = pd.DataFrame(selector.fit_transform(X_encoded), columns=X_encoded.columns[selector.get_support()])

print("\n--- AFTER FEATURE SELECTION (low-variance features removed) ---")
print(X_selected.sample(10, random_state=42))

# Train-test split on the selected features
X_train, X_test, y_train, y_test = train_test_split(X_selected, y, test_size=0.2, random_state=42)

# Train a model directly on the prepared data
clf = LogisticRegression(max_iter=1000)
clf.fit(X_train, y_train)

# Evaluate
scores = cross_val_score(clf, X_selected, y, cv=5, scoring='accuracy')
print(f"\nCross-validation accuracy (on prepared data): {scores.mean():.3f}")
```

```
--- AFTER FEATURE ENCODING (categorical → numeric) ---
           age     sibsp     parch      fare  pclass_1  pclass_2  pclass_3  \
1148  0.426099 -0.479087 -0.445000 -0.505708       0.0       0.0       1.0
1049 -0.736663  0.481288  0.710763 -0.339111       0.0       0.0       1.0
982  -0.116523 -0.479087 -0.445000 -0.490805       0.0       0.0       1.0
808  -0.116523 -0.479087 -0.445000 -0.487824       0.0       0.0       1.0
1195 -0.116523 -0.479087 -0.445000 -0.493624       0.0       0.0       1.0
240   1.201274 -0.479087 -0.445000 -0.130140       1.0       0.0       0.0
1118 -0.349075 -0.479087 -0.445000 -0.490240       0.0       0.0       1.0
596   0.116029 -0.479087 -0.445000 -0.392119       0.0       1.0       0.0
924   0.387341 -0.479087 -0.445000 -0.492093       0.0       0.0       1.0
65    0.271064  0.481288 -0.445000  0.383183       1.0       0.0       0.0

      sex_female  sex_male  embarked_C  embarked_Q  embarked_S
1148         0.0       1.0         0.0         0.0         1.0
1049         0.0       1.0         1.0         0.0         0.0
982          0.0       1.0         0.0         0.0         1.0
808          0.0       1.0         0.0         0.0         1.0
1195         0.0       1.0         0.0         1.0         0.0
240          0.0       1.0         0.0         0.0         1.0
1118         0.0       1.0         0.0         0.0         1.0
596          0.0       1.0         0.0         0.0         1.0
924          0.0       1.0         0.0         1.0         0.0
65           1.0       0.0         0.0         0.0         1.0
```

Stage	Purpose	Mathematical Aspect	Transformation Example
Feature Encoding	Map categorical data to numeric vectors	One-Hot Encoding $\rightarrow \mathbb{R}^n$	sex=female → [1,0]
Feature Selection	Remove low-variance (uninformative) features	$Var(X_j) < \varepsilon \Rightarrow \text{drop}$	drops constants

```
--- AFTER FEATURE SELECTION (low-variance features removed) ---
           age     sibsp     parch      fare  pclass_3  sex_female  sex_male
1148  0.426099 -0.479087 -0.445000 -0.505708       1.0         0.0       1.0
1049 -0.736663  0.481288  0.710763 -0.339111       1.0         0.0       1.0
982  -0.116523 -0.479087 -0.445000 -0.490805       1.0         0.0       1.0
808  -0.116523 -0.479087 -0.445000 -0.487824       1.0         0.0       1.0
1195 -0.116523 -0.479087 -0.445000 -0.493624       1.0         0.0       1.0
240   1.201274 -0.479087 -0.445000 -0.130140       0.0         0.0       1.0
1118 -0.349075 -0.479087 -0.445000 -0.490240       1.0         0.0       1.0
596   0.116029 -0.479087 -0.445000 -0.392119       0.0         0.0       1.0
924   0.387341 -0.479087 -0.445000 -0.492093       1.0         0.0       1.0
65    0.271064  0.481288 -0.445000  0.383183       0.0         1.0       0.0

Cross-validation accuracy (on prepared data): 0.730
```

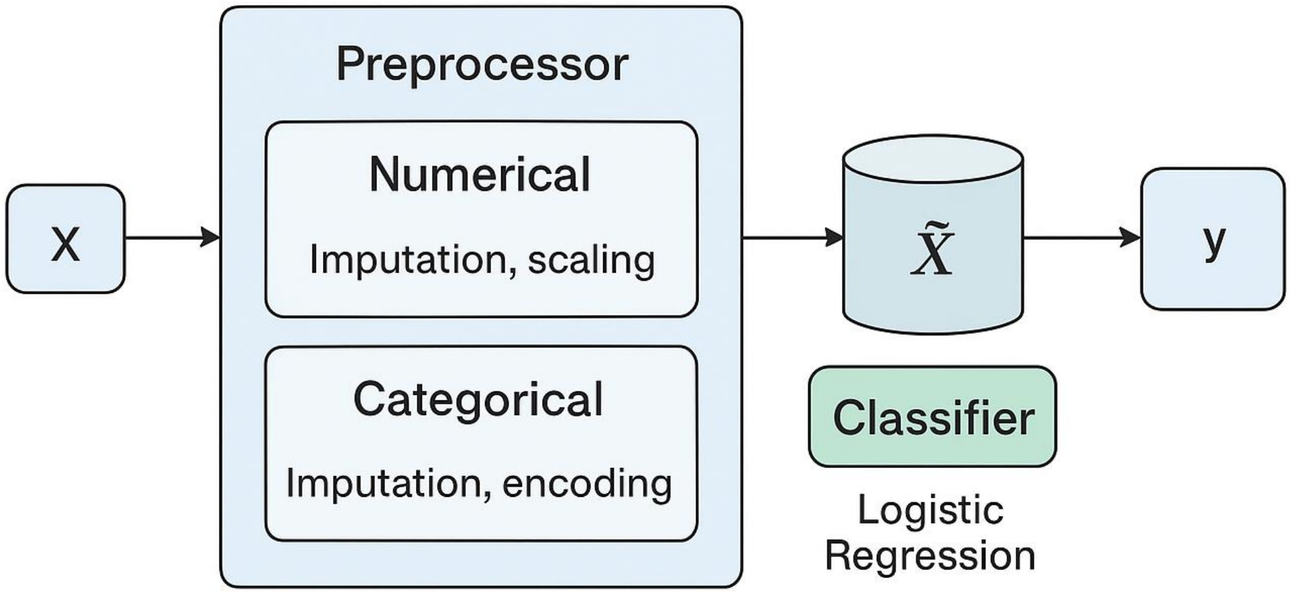
Building an ML Pipeline (Integration of All Steps)

Class / Function	Description	Key Parameters	Example Usage
sklearn.pipeline.Pipeline	Sequentially connects preprocessing and modeling steps into one composite estimator. Each step executes in order.	steps=[('scaler', StandardScaler()), ('model', LogisticRegression())], verbose	pipe = Pipeline([...]); pipe.fit(X_train, y_train)
sklearn.compose.ColumnTransformer	Applies different transformations to different columns (e.g., scaling for numerical, encoding for categorical).	transformers=[('num', scaler, num_cols), ('cat', encoder, cat_cols)], remainder='drop'	ct = ColumnTransformer([...])
sklearn.model_selection.cross_val_score	Evaluates entire pipeline performance via cross-validation.	cv=5, scoring='accuracy'	cross_val_score(pipe, X, y, cv=5)
sklearn.metrics (e.g. accuracy_score, classification_report)	Evaluates pipeline output on test data.	—	y_pred = pipe.predict(X_test); accuracy_score(y_test, y_pred)
sklearn.pipeline.make_pipeline	Simplified factory method for creating pipelines without naming steps explicitly.	*steps	pipe = make_pipeline(StandardScaler(), SVC())

Key Concepts

1. A **Pipeline** guarantees that all transformations are applied consistently during both training and inference.
2. It simplifies **cross-validation** and **hyperparameter tuning** — all preprocessing happens inside each fold.
3. You can include preprocessing (scaling, encoding, feature selection), anomaly handling, and the model in a single workflow.
4. It ensures **no data leakage** — transformations are fit only on the training portion.

Complete Code Example — Full Titanic Data Pipeline




```python
import pandas as pd
import numpy as np
from sklearn.model_selection import train_test_split, cross_val_score
from sklearn.preprocessing import StandardScaler, OneHotEncoder
from sklearn.impute import SimpleImputer
from sklearn.compose import ColumnTransformer
from sklearn.pipeline import Pipeline
from sklearn.linear_model import LogisticRegression
from sklearn.datasets import fetch_openml
from sklearn.feature_selection import VarianceThreshold

# === 1. Load Titanic dataset ===
X, y = fetch_openml("titanic", version=1, as_frame=True, return_X_y=True)
y = y.astype(int)

print("\n--- INITIAL DATA (10 random samples) ---")
print(X.sample(10, random_state=42))
```

```
--- INITIAL DATA (10 random samples) ---
      pclass                                             name     sex   age  sibsp  parch             ticket     fare cabin embarked boat  body                        home.dest
1148       3                           Rintamaki, Mr. Matti    male  35.0      0      0  STON/O 2. 3101273   7.1250   NaN        S  NaN   NaN                              NaN
1049       3                              Nakid, Mr. Sahid    male  20.0      1      1               2653  15.7417   NaN        C    C   NaN                              NaN
982        3                          Lyntakoff, Mr. Stanko    male   NaN      0      0             349235   7.8958   NaN        S  NaN   NaN                              NaN
808        3                              Ford, Mr. Arthur    male   NaN      0      0          A/5 1478   8.0500   NaN        S  NaN   NaN  Bridgwater, Somerset, England
1195       3                       Shaughnessy, Mr. Patrick    male   NaN      0      0             370374   7.7500   NaN        Q  NaN   NaN                              NaN
240        1   Romaine, Mr. Charles Hallace ('Mr C Rolmane')    male  45.0      0      0             111428  26.5500   NaN        S    9   NaN                    New York, NY
1118       3                 Peltomaki, Mr. Nikolai Johannes    male  25.0      0      0  STON/O 2. 3101291   7.9250   NaN        S  NaN   NaN                              NaN
596        2                          Wilhelms, Mr. Charles    male  31.0      0      0             244270  13.0000   NaN        S    9   NaN                  London, England
924        3                              Kelly, Mr. James    male  34.5      0      0             330911   7.8292   NaN        Q  NaN  70.0                              NaN
65         1  Chambers, Mrs. Norman Campbell (Bertha Griggs)  female  33.0      1      0             113806  53.1000    E8        S    5   NaN  New York, NY / Ithaca, NY
```

Complete Code Example — Full Titanic Data Pipeline

```
# === 2. Create full pipeline ===
num_transformer = Pipeline(steps=[
    ('imputer', SimpleImputer(strategy='median')),
    ('scaler', StandardScaler())
])

cat_transformer = Pipeline(steps=[
    ('imputer', SimpleImputer(strategy='most_frequent')),
    ('encoder', OneHotEncoder(handle_unknown='ignore'))
])

preprocessor = ColumnTransformer(transformers=[
    ('num', num_transformer, num_features),
    ('cat', cat_transformer, cat_features)
])

model = Pipeline(steps=[
    ('preprocessor', preprocessor),
    ('selector', VarianceThreshold(threshold=0.22)),
    ('classifier', LogisticRegression(max_iter=1000))
])

# === 3. Train-test split & fit model ===
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)
model.fit(X_train, y_train)

# === 4. Evaluate ===
scores = cross_val_score(model, X, y, cv=5, scoring='accuracy')
print(f"\nCross-validation accuracy: {scores.mean():.3f}")
```

```
Cross-validation accuracy: 0.734
```

Predict the survival of a new (random) passenger:

```
# === 5. Create random passenger ===
# (Using reasonable ranges based on the Titanic dataset)
random_passenger = pd.DataFrame([{
    "pclass": np.random.choice(["1st", "2nd", "3rd"]),
    "sex": np.random.choice(["male", "female"]),
    "age": np.random.uniform(1, 70),
    "sibsp": np.random.randint(0, 3),
    "parch": np.random.randint(0, 2),
    "fare": np.random.uniform(10, 150),
    "embarked": np.random.choice(["C", "Q", "S"])
}])

print("\nRandom passenger profile:")
print(random_passenger)

# === 6. Predict survival ===
prediction = model.predict(random_passenger)[0]
prob = model.predict_proba(random_passenger)[0, 1]

# === 7. Output result ===
print(f"\nPredicted survival: {'Yes' if prediction == 1 else 'No'} (probability = {prob:.2f})")
```

```
Random passenger profile:
  pclass   sex        age  sibsp  parch      fare embarked
0    1st  male  34.561246      0      1  46.81821        S

Predicted survival: No (probability = 0.28)
```

```
Random passenger profile:
  pclass     sex        age  sibsp  parch       fare embarked
0    3rd  female  41.128766      1      0  89.419109        C

Predicted survival: Yes (probability = 0.82)
```

From Raw Data to Prediction — Key Concepts

Stage	Tool / Concept	Purpose in the ML Workflow
1. Data Understanding	pandas, info(), describe()	Explore structure, detect missing values, identify data types.
2. Data Cleaning	**Imputer** (SimpleImputer)	Fill or remove missing data consistently.
3. Feature Preparation	**Encoder** (OneHotEncoder), **Scaler** (StandardScaler)	Convert categories and scale numerical features for model readiness.
4. Feature Selection / Reduction	Filter methods, PCA	Keep the most informative features, reduce dimensionality and noise.
5. Transformation Control	**ColumnTransformer**	Apply different transformations to specific feature groups.
6. Workflow Automation	**Pipeline**	Combine preprocessing + model training into a single repeatable process.
7. Model Training	LogisticRegression, RandomForest, etc.	Learn relationships from training data.
8. Validation & Testing	**Cross-validation**, train_test_split	Evaluate model performance and generalization.
9. Real-world Use	predict() / predict_proba()	Make predictions for new (unseen) samples.

A good ML model is not just a classifier — it's a *reproducible pipeline* that transforms raw data into **reliable predictions**

Let’s proceed to the team project