2.1 - Data Pre-Processing

Giovanni Della Lunga
giovanni.dellalunga@unibo.it

Introduction to Machine Learning for Finance

Bologna - February-March, 2025

Why Data Preprocessing Is Central in Finance

In financial machine learning, preprocessing is not a preliminary step. It is part of the economic and statistical model.

Seemingly harmless preprocessing choices can:

- introduce bias,
- distort risk and tail behavior,
- invalidate backtests,
- destroy out-of-sample performance.

This lesson explains why preprocessing decisions matter at least as much as model choice.

Learning Objectives

At the end of this lecture, you should be able to:

- Understand preprocessing as a modeling decision, not a technical fix.
- Critically analyze when missing data is informative rather than accidental.
- Choose encoding strategies consistent with economic meaning.
- Apply scaling methods that respect time, risk, and market structure.

Goal: move from "cleaning data" to "modeling the data-generating process".

Didactic Targets

This lecture is designed to develop:

- **Conceptual awareness**: recognize hidden assumptions in preprocessing.
- **Financial intuition**: connect data issues to liquidity, risk, and behavior.
- **Critical judgment**: question default pipelines and “standard practices”.

You are not learning recipes. You are learning to reason.

What This Lecture Will *Not* Do

This lecture will not:

- provide universal preprocessing recipes,
- claim that one method is always optimal,
- treat financial data as generic tabular data,

In finance, there are no preprocessing shortcuts.

Introduction

"DATA! DATA! DATA!
i can't make
bricks without clay!"
- Sir Arthur Conan Doyle

Introduction
Data Pre-Processing

- Before applying any machine learning model, it is essential to establish a rigorous understanding of **data pre-processing**.
- Poorly prepared data can lead to misleading conclusions, rendering even the most sophisticated algorithms ineffective.
- Data pre-processing introduces several key concepts that are **transversal to all of machine learning**; among these, **handling missing values**, **feature scaling**, **encoding categorical variables**, and **detecting outliers** are crucial.
- These techniques ensure that the input data is structured, consistent, and suitable for training robust models.
- Without proper data preparation, models may struggle with convergence, exhibit bias, or fail to generalize to new data.

Introduction

Data Pre-Processing

In this lesson, we will cover the essential components of data pre-processing, including:

- Handling missing data: imputation and removal strategies
- Feature scaling: normalization and standardization
- Encoding categorical variables
- Detecting and handling outliers
- Feature Selection and Dimensionality Reduction

Data Pre-Processing

- Raw data rarely comes in the form and shape that is necessary for the optimal performance of a learning algorithm.
- On the other hand, the success of a machine learning algorithm highly depends on the quality of the data fed into the model.
- Real-world data is often dirty containing outliers, missing values, wrong data types, irrelevant features, or non-standardized data.
- The presence of any of these will prevent the machine learning model to properly learn.
- For this reason, transforming raw data into a useful format is an essential stage in the machine learning process.

Data Cleaning

- Dealing with inconsistent recording
- Removing unwanted observations
- Removing duplicates
- Investigating outliers
- Dealing with missing items

Missing Data

Handling Missing Data

- The real-world data often has a lot of missing values.
- The cause of missing values can be data corruption or failure to record data.
- The handling of missing data is very important during the preprocessing of the dataset as many machine learning algorithms do not support missing values.

Handling Missing Data

Delete Rows/Columns with Missing Values

- One of the easiest ways to deal with missing data is simply to remove the corresponding features (columns) or training examples (rows) from the dataset entirely;
- Missing values can be handled by deleting the rows or columns having null values. If columns have more than half of the rows as null then the entire column can be dropped. The rows which are having one or more columns values as null can also be dropped.
- Remember that, in pandas, rows with missing values can easily be dropped via the **dropna** method.

Handling Missing Data

Delete Rows/Columns with Missing Values

Pros:

- A model trained with the removal of all missing values creates a robust model.

Cons:

- Loss of a lot of information.
- Works poorly if the percentage of missing values is excessive in comparison to the complete dataset.

Handling Missing Data

Imputing missing values with Mean/Median:

- Columns in the dataset which have numeric continuous values can be replaced with the mean, median, or mode of remaining values in the column.
- This method can prevent the loss of data compared to the earlier method.

Handling Missing Data

Imputing missing values with Mean/Median

Pros:

- Prevent data loss which results in deletion of rows or columns
- Works well with a small dataset and is easy to implement.

Cons:

- Works only with numerical continuous variables.
- Can cause data leakage
- Do not factor the covariance between features.

Missing Data in Finance: Why It Matters

In financial datasets, missing values are rarely an innocuous technical detail. They often reflect **economic mechanisms** (illiquidity, reporting choices, market microstructure) and can introduce:

- **Bias** (selection effects, survivorship bias)
- **Distorted risk estimates** (volatility and tail risk attenuation)
- **Look-ahead / information leakage** (improper use of future information)

Key message: **missingness is part of the data-generating process**.

Example: Mean Imputation Distorts Volatility

Observed returns:

$$[0.01,\ \mathrm{NaN},\ -0.04,\ \mathrm{NaN},\ 0.03]$$

Mean of observed values: $\mu = 0.0$

After mean imputation:

$$[0.01,\ 0.00,\ -0.04,\ 0.00,\ 0.03]$$

Effect:

- variance decreases,
- tail risk is attenuated,
- volatility is artificially smoothed.

Mean imputation hides risk exactly where finance cares most.

Mechanisms of Missingness: MCAR, MAR, MNAR

A principled treatment starts from the missingness mechanism:

- **MCAR** (Missing Completely At Random): missingness independent of data
- **MAR** (Missing At Random): missingness depends on observed variables
- **MNAR** (Missing Not At Random): missingness depends on unobserved values

In finance, **MNAR is common**:

- Firms stop reporting when distressed
- Illiquid assets have stale/missing prices when volatility is high
- Coverage gaps reflect risk and attention, not randomness

Typical Finance Examples of Missing Data

Market data

- Missing trades/quotes due to **illiquidity** or **market closures**
- Stale prices (last observation carried forward) during low activity

Fundamentals / accounting

- Quarterly reporting lags, restatements, missing line items
- Missingness correlated with firm size, leverage, distress (often MNAR)

Credit and alternative data

- Missing ratings, missing features for thin-file applicants
- Platform policy changes (structural breaks)

Deletion Is Not Neutral: Selection and Survivorship Bias

Dropping rows/columns with missing values may be convenient but can be dangerous:

- **Selection bias**: removing distressed firms changes default/risk distribution
- **Survivorship bias**: datasets become biased toward persistent/healthy entities
- **Regulatory concerns**: biased credit decisions may violate fairness requirements

Rule of thumb:

If missingness is informative (MAR/MNAR), deletion changes the target population.

Imputation: Benefits, Pitfalls, and Leakage

Simple imputations (mean/median) can:

- reduce variance and **artificially smooth** returns/volatility
- destroy cross-feature dependence and time structure

Information leakage risk:

- Computing imputation statistics on the full dataset uses future information
- In finance, time ordering is a **causal constraint**

Best practice in ML pipelines:

- Fit imputers **only on training data**
- Use **time-based splits** (no random shuffling for time series)

Missingness as a Signal: Add Indicators

A **missingness indicator** is an auxiliary feature that explicitly tells the model whether a value was originally missing.

- It does **not** try to “fix” missing data by itself.
- It makes the **fact of missingness** visible to the learning algorithm.

Key idea:

In many financial datasets, **missingness carries information**.

Why It Makes Sense in Finance (Economic Intuition)

In finance, missing values are often **not random**. The reason a value is missing may reflect:

- **Liquidity and market microstructure** (no trades, stale quotes)
- **Disclosure and reporting behavior** (firms omit items strategically)
- **Risk and transparency** (thin-file borrowers, incomplete applications)

Therefore, missingness can be a **signal** about the data-generating process.

What a Missingness Indicator Does (Technically)

Given a feature x, we create a binary indicator m:

$$m = \begin{cases} 1 & \text{if } x \text{ is missing} \\ 0 & \text{if } x \text{ is observed} \end{cases}$$

Then the model receives:

- an imputed value for x (so no NaNs remain),
- and the indicator m (so the model can treat imputed values differently).

Understanding the Notation in the Formula

We write:

$$m_{i,t} = \mathbf{1}\{x_{i,t} \text{ is missing}\}.$$

- $x_{i,t}$: value of feature x for entity i (asset/firm/client) at time t.
- $\mathbf{1}\{\cdot\}$: **indicator function**:

$$\mathbf{1}\{A\} = \begin{cases} 1 & \text{if event } A \text{ is true} \\ 0 & \text{otherwise} \end{cases}$$

So $m_{i,t} = 1$ exactly when $x_{i,t}$ was not observed.

Example 1: Financial Time Series

Suppose we observe a price series with missing values:

Date	Price
t_1	100
t_2	NaN
t_3	NaN
t_4	105

After time-consistent imputation (e.g., forward fill) and adding the indicator:

Date	$\text{Price}_{\text{imp}}$	$\text{Price}_{\text{miss}}$
t_1	100	0
t_2	100	1
t_3	100	1
t_4	105	0

The model can learn: "imputed prices are less reliable".

Example 2: Manual Python Implementation

```
import pandas as pd
import numpy as np

df = pd.DataFrame({"leverage": [0.4, np.nan, 0.7, np.nan, 0.5]})

# missingness indicator
df["leverage_missing"] = df["leverage"].isna().astype(int)

# imputation (median)
df["leverage_imp"] = df["leverage"].fillna(df["leverage"].median())
```

Result: the model receives both `leverage_imp` and `leverage_missing`.

A Proper Sklearn Pipeline Implementation

Scikit-learn can add indicators automatically (fit on train only):

```
from sklearn.impute import SimpleImputer

imputer = SimpleImputer(strategy="median", add_indicator=True)

X_train_imp = imputer.fit_transform(X_train)
X_test_imp  = imputer.transform(X_test)
```

- The imputer learns statistics from **training data only**.
- Indicators are appended for features that had missing values.
- This helps prevent **information leakage**.

When It Works Well (and When It Does Not)

Works well when:

- missingness is informative (often MAR/MNAR in finance),
- models can exploit interactions (trees, boosting, neural nets),
- panel datasets: many entities over time.

Less useful when:

- missingness is truly random (MCAR),
- very rigid linear models without interactions,
- missingness is extremely rare and irrelevant.

Takeaway Message

In finance, missing data is often a **feature**, *not a bug.*

Practical rule:

- Impute in a **time-consistent** way,
- add missingness indicators,
- let the model decide whether missingness is predictive.

This approach is often more robust than hiding missing values with a single constant.

Time-Aware Imputation Strategies (Finance)

For time series and panel data, prefer time-consistent strategies:

- **Forward fill** (with limits): appropriate for stale quotes, but must cap staleness
- **Rolling statistics** (mean/median): computed with past window only
- **Model-based imputation**: predict missing values from observed covariates (train-only)
- **Cross-sectional imputation**: within-date imputation using peers/industry groups

Important: choose the method consistent with the **economic meaning** of the feature.

A Practical Workflow for Missing Data in Finance

Recommended workflow:

1. **Quantify missingness** by feature, time, and entity (issuer/asset)
2. **Diagnose mechanism**: MCAR vs MAR vs MNAR (domain + tests)
3. **Decide strategy**:
   - keep + indicator if missingness informative
   - time-aware imputation if measurement gaps are technical
   - deletion only if missingness is plausibly MCAR and small
4. **Prevent leakage**: fit all preprocessing on training only, time-split validation

Takeaways

- In finance, missing data is often **informative**, not accidental.
- Deletion can induce **selection/survivorship bias**.
- Simple imputations can distort **risk, tails, and dependence**.
- Use **time-aware pipelines**, and consider **missingness indicators**.

Categorical Data

A Quick Remind about Data Types

- Machine learning handles various data types: numerical, categorical, ordinal, and textual.
- Proper preprocessing (e.g., encoding, normalization) is essential.
- Choosing the right model depends on the type of data available.

Numerical Data

- **Definition:** Numerical data represents measurable quantities.
- **Types:**
  - **Continuous** - Can take any real value (e.g., height, weight, temperature).
  - **Discrete** - Countable values (e.g., number of transactions, age in years).
- **Example:** House prices, stock prices.

Categorical Data

- **Definition:** Categorical data represents discrete groups or labels.
- **Types:**
    - **Nominal** - No natural order (e.g., colors, gender, country names).
    - **Ordinal** - Has an order but no fixed distance (e.g., education level, customer ratings).
- **Example:** Car brands, job titles.

Textual Data

- **Definition:** Data in the form of text or language-based information.
- **Examples:** Reviews, social media posts, customer feedback.
- **Processing Methods:** Natural Language Processing (NLP), tokenization, embeddings.

What is Categorical Data?

Definitions

- Categorical data refers to data that consists of discrete labels or categories rather than numerical values.
- These categories represent different groups or classifications, such as colors (red, blue, green), gender (male, female, non-binary), or product types (electronics, clothing, furniture).
- There are two types of categorical variables:

  - **I. Ordinal Variables**

  - **II. Nominal Variables**

Categorical Data: Encoding

- Most of the Machine learning algorithms are designed to work with numeric data.
- Hence, we need to convert Categorical (text) data into numerical data for model building.
- There are multiple encoding techniques to convert categorical data into numerical data. Let us look at some of them.

Categorical Encoding Is a Modeling Choice

Categorical encoding is often presented as a technical preprocessing step. **In financial applications, this can be misleading**.

> *Encoding a categorical variable means making* **explicit assumptions** *about structure, distance, and economic meaning.*

Different encodings of the same variable can lead to:

- different decision boundaries,
- different risk estimates,
- different out-of-sample behavior.

Financial Categorical Variables: Not All Are Alike

Typical categorical variables in finance include:

- **Credit ratings** (AAA, AA, A, . . .)
- **Industry / sector classifications**
- **Country of risk / domicile**
- **Contract types, product classes, regimes**

Key question:

> *Does this category encode an* **ordering**, *a* **group identity**, *or an* **economic state**?

The answer determines the appropriate encoding.

Subsection 1

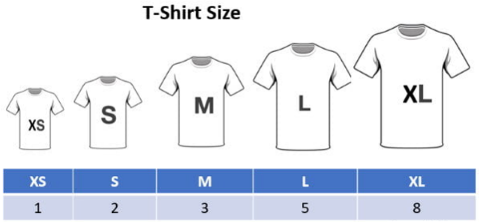
Categorical Data

Ordinal Encoding

Categorical Data: Ordinal Encoding

Ordinal Variables

- **Ordinal Encoding** primarily encodes ordinal categories into ordered numerical values.
- Ordinal encoding maps each unique category value to a specific numerical value based on its order or rank.
- These variables maintain a natural order in their class of values.



XS	S	M	L	XL
1	2	3	5	8

Ordinal Encoding in Finance
A Strong Structural Assumption

Ordinal encoding assumes:

- categories can be totally ordered,
- distances between levels are **meaningful to the model**.

Example: Credit ratings

$$\text{AAA} > \text{AA} > \text{A} > \text{BBB} > \dots$$

Implicit assumption:

The model treats differences between adjacent ratings as comparable.

This is **not a neutral choice**, but an economic hypothesis.

When Ordinal Encoding Makes Sense in Finance

Ordinal encoding is reasonable when:

- the ordering reflects a **latent continuous variable** (e.g., creditworthiness),
- the downstream model can handle **nonlinear effects** (trees, boosting),
- interpretability is aligned with economic intuition.

Typical use cases:

- credit scoring with tree-based models,
- stress testing with ordered risk classes,
- regulatory-style scorecards.

When Ordinal Encoding Is Dangerous

Ordinal encoding can be problematic when:

- used with **linear or distance-based models**,
- the true relationship is non-monotonic,
- category spacing is arbitrary or unstable over time.

Risk:

The model interprets numeric distances as economic distances.

This can lead to:

- spurious linear effects,
- false extrapolation,
- unstable coefficients out-of-sample.

Ordinal Encoding - Python Code

Python Implementation:

```
from sklearn.preprocessing import OrdinalEncoder

ratings = [["AAA", "AA", "A", "BBB", "BB", "B", "CCC", "CC", "C", "D"]]

le = OrdinalEncoder(categories=ratings)
df["rating_encoded"] = le.fit_transform(df[["rating"]])
df.head()
```

Subsection 2

Categorical Data

Nominal Encoding

Nominal Variables: Group Identity, Not Order

Nominal categorical variables encode **membership**, not ranking.
Examples:

- industry sector,
- country,
- exchange venue,
- contract type.

There is no meaningful notion of:

$$\text{Technology} > \text{Utilities}$$

Any encoding that introduces order here is conceptually wrong.

Categorical Data: Label Encoding

- Consider for example a target variable 'Job Status' that has four different categories.
- After applying label encoding to this column the four different categories are mapped into integers 0: Full Time, 1: Intern, 2: Part-Time, and 3:Unemployed.
- With this, it can be interpreted that Unemployed have a higher priority than Part-Time, Full Time, and Intern while training the model, whereas, **there is no such priority or relation between these statuses**.
- We cannot define the order of labels with the label encoding technique.

Comparison Between Ordinal and Label Encoding

- Both **Ordinal Encoding** and **Label Encoding** convert categorical data into numbers.
- However, they serve **different purposes** and should not be used interchangeably.

Ordinal Encoding

Comparison Between Ordinal and Label Encoding

Concept:

- Assigns an integer to each category **based on a predefined order**.
- **Preserves order** in the data.

Example: Credit Ratings

Rating	**Ordinal Encoding**
CCC	0
B	1
BB	2
BBB	3
A	4
AA	5
AAA	6

Label Encoding
Comparison Between Ordinal and Label Encoding

Concept:

- Assigns a unique integer to each category, but **without any ordering**.
- Numbers **do not represent a ranking**.

Example: Car Brands

Car Brand	Label Encoding
Toyota	0
Ford	1
BMW	2
Tesla	3

Label Encoding - Python Code

Comparison Between Ordinal and Label Encoding

Python Implementation:

```
from sklearn.preprocessing import LabelEncoder

data = pd.DataFrame({'car_brand': ['Toyota', 'Ford',
                     'BMW', 'Tesla', 'Ford']})
encoder = LabelEncoder()
data['brand_encoded'] = encoder.fit_transform(data['car_brand'])
print(data)
```

Key Differences

Comparison Between Ordinal and Label Encoding

Comparison Table

Feature	**Ordinal Encoding**	**Label Encoding**
Preserves Order?	Yes	No
Use Case	Ordered Categories	Unordered
Assigns Numbers?	Yes	Yes
Represents Ranking?	Yes	No
Risk of Misinterpretation?	If order is wrong	If used on ordered data
Scikit-Learn Class	OrdinalEncoder	LabelEncoder

Conclusions

Comparison Between Ordinal and Label Encoding

- Use **Ordinal Encoding** only when categories have a meaningful order.
- Use **Label Encoding** for nominal data where order does not matter.
- Consider alternative encoding methods (e.g., **One-Hot Encoding**) for categorical variables with many unique values.

Categorical Data: One Hot Encoding

What is One Hot Encoding?

- if there is no ordinal relationship between the categorical variables then ordinal encoding might mislead the model.
- This is because the ordinal encoder will try to force an ordinal relationship on the variables to assume a natural ordering, thus resulting in poor performance.
- In this case, One Hot encoder should be used to treat our categorical variables.
- It will create dummy variables by converting N categories into N features/columns.

Original Binary Variable
Categorical Data: One Hot Encoding

A binary variable example that illustrates one-hot encoding is "Has Mortgage", where applicants either have a mortgage (`Yes`) or do not (`No`).

Applicant ID	Has Mortgage
001	Yes
002	No
003	Yes
004	No

Table: Original Binary Variable

After One-Hot Encoding

Categorical Data: One Hot Encoding

A binary variable example that illustrates one-hot encoding is "Has Mortgage", where applicants either have a mortgage (`Yes`) or do not (`No`).

Applicant ID	**Has Mortgage (Yes)**	**Has Mortgage (No)**
001	1	0
002	0	1
003	1	0
004	0	1

Table: One-Hot Encoded Binary Variable

After One-Hot Encoding

Categorical Data: One Hot Encoding

We can one-hot encode categorical variables in two ways: by using **get_dummies** in pandas and by using **OneHotEncoder** from sklearn.

Applicant ID	Has Mortgage (Yes)	Has Mortgage (No)
001	1	0
002	0	1
003	1	0
004	0	1

Table: One-Hot Encoded Binary Variable

One-Hot Encoding: Safe but Not Free

One-Hot Encoding avoids artificial ordering by creating indicator variables.
Advantages:

- no imposed ranking,
- model can learn category-specific effects,
- compatible with linear models.

Costs (especially in finance):

- high dimensionality,
- sparsity,
- instability with rare categories.

High-Cardinality Categories in Finance

Financial datasets often contain categories with many levels:

- issuers,
- instruments,
- counterparties,
- tickers.

Naive one-hot encoding can:

- explode dimensionality,
- overfit to idiosyncratic entities,
- destroy generalization.

In these cases, encoding choices strongly affect model validity.

Takeaways for Financial Applications

- Encoding is not a technical detail but a **modeling assumption**.
- The same categorical variable can require different encodings depending on the task.
- Always ask:

  What economic structure am I imposing by this encoding?

- In finance, wrong encoding can be worse than no encoding.

Feature Scaling (Normalization)

Feature Scaling Is Not Neutral in Finance

- Before using many ML algorithms (including those for unsupervised learning), it is important to scale feature values so that they are comparable.
- Feature scaling is often presented as a numerical convenience. In financial applications, this is misleading.

  Scaling transforms the geometry of the data and implicitly defines what "large" and "small" risks mean.

Different scaling choices can change:

- relative importance of features,
- model stability over time,
- interpretation of coefficients and signals.

Why Scaling Matters in Financial Models

Financial features often differ in:

- units (returns, prices, volumes, ratios),
- scale (basis points vs millions),
- volatility and tail behavior.

Without scaling:

- distance-based models become meaningless,
- optimization may be ill-conditioned,
- some features dominate purely due to units.

Scaling is therefore necessary — but never innocuous.

Z-Score Scaling: Implicit Assumptions

Standardization (Z-score):

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

Implicit assumptions:

- existence of finite mean and variance,
- approximate stationarity,
- symmetry around the mean.

In finance, these assumptions are often violated:

- heavy tails,
- regime changes,
- volatility clustering.

Feature Scaling

- Min-max scaling involves calculating the maximum and minimum value of each feature from the training set.
- Scaled feature values for all data sets are then created by subtracting the minimum and dividing by the difference between the maximum and minimum.
- The scaled feature values lie between zero and one:

$$\text{scaled feature value} = \frac{V - V_{min}}{V_{max} - V_{min}} \tag{1}$$

Min–Max Scaling: When It Breaks

Min–max scaling maps values to $[0, 1]$ using:

$$x' = \frac{x - x_{\min}}{x_{\max} - x_{\min}}$$

Problems in finance:

- extreme sensitivity to outliers,
- unstable bounds in expanding samples,
- implicit use of future information if not time-aware.

Min–max scaling is rarely appropriate for financial time series.

Time-Series vs Cross-Sectional Scaling

In finance, scaling depends on the data dimension:

- **Time-series scaling**: normalize each asset over time
- **Cross-sectional scaling**: normalize across assets at a given date

These operations answer different questions:

- Time-series: “Is this value unusual for this asset?”
- Cross-sectional: “How does this asset compare to its peers today?”

Cross-Sectional Scaling in Asset Pricing

Cross-sectional scaling is standard in factor models:

- rank or Z-score characteristics across assets at time t,
- remove market-wide level effects,
- focus on relative signals.

Typical workflow:

- compute feature values at time t,
- scale across assets,
- use scaled values for portfolio construction or prediction.

Rolling and Time-Aware Scaling

To avoid look-ahead bias, scaling must respect time ordering.
Best practices:

- compute mean and variance using rolling windows,
- update scaling parameters only with past data,
- align window length with investment horizon.

This makes scaling a **dynamic transformation**, not a static one.

Scaling, Risk, and Interpretability

Scaling choices affect economic interpretation:

- Z-score scaling links signals to deviations from “normal” behavior,
- cross-sectional ranks emphasize relative mispricing,
- unscaled variables preserve absolute economic meaning.

Question to ask:

Do I want my model to react to absolute levels or relative anomalies?

Scaling Depends on the Model Class

Different models respond differently to scaling:

Model Type	**Scaling Needed?**
Linear / Regularized	Yes (interpretability, stability)
Tree-based	Often no (but helpful for consistency)
Distance-based (kNN, SVM)	Essential
Neural Networks	Essential

In finance, scaling choices interact with both model and data structure.

Key Takeaways

- Feature scaling encodes assumptions about risk and comparability.
- Z-score and min–max scaling impose strong statistical hypotheses.
- Time-aware and cross-sectional scaling are central in finance.
- Incorrect scaling can invalidate otherwise sound models.

Conclusions

Key Takeaways: Missing Data

- Missing data in finance is often informative, not random.
- Deleting observations can introduce selection and survivorship bias.
- Simple imputation can smooth volatility and hide tail risk.
- Missingness indicators allow models to exploit economic signals.
- Time-aware preprocessing is essential to avoid leakage.

Missing data is part of the data-generating process.

Key Takeaways: Categorical Encoding

- Encoding is a structural modeling choice, not a technical detail.
- Ordinal encoding imposes assumptions about distances and order.
- Nominal categories encode identity, not ranking.
- Wrong encoding can create spurious relationships.
- In finance, encoding mistakes are often worse than no encoding.

Always ask: what economic structure am I imposing?

Key Takeaways: Feature Scaling

- Scaling defines what “large” and “small” mean for the model.
- Z-score and min–max scaling embed strong statistical assumptions.
- Time-series and cross-sectional scaling answer different questions.
- Rolling and time-aware scaling reduce look-ahead bias.
- Scaling choices affect interpretability and stability.

Scaling is part of the economic model.

Overall Takeaways

- Preprocessing is inseparable from modeling in finance.
- Every preprocessing choice encodes assumptions.
- Many financial ML failures originate in the data pipeline.
- Backtest-safe preprocessing is a necessary condition for success.
- Critical thinking beats automated pipelines.

In financial machine learning, bad preprocessing can invalidate even the best model.

Looking Ahead

In the next lecture we will see that:

- TBD

Data, preprocessing, and targets form a single modeling pipeline.