

# Data Preparation I (Cleaning & Preprocessing)

Prepared by Dr. Sumaya Alkubaisi

FNT633

Machine Learning in Finance

Lecture 2



# Learning Goals

- By the end of this lecture, students will be able to:
- Understand data cleaning & preprocessing concepts
- Retrieve data from CSV, JSON, SQL databases, APIs
- Handle missing values, outliers, and inconsistent data
- Implement cleaning techniques in Python Jupyter

# Why Data Preparation Matters

## 1. Garbage in → Garbage out

- Machine learning models can only be as good as the data they learn from.
- If the data is messy, incomplete, or wrong, the model will make poor predictions.
- Think of it like cooking: if you use spoiled ingredients, the dish will not taste good no matter how skilled the chef is.

## 2. Most of the work is in preparing the data

- Around 80% of the effort in a machine learning project goes into cleaning and organizing data.
- This includes removing errors, filling missing values, standardizing formats, and making sure the data is consistent.
- Skipping this step can make all the time spent on modeling useless.

# Why Data Preparation Matters

Example: Loan dataset

Suppose we have a dataset of loan applicants, but many entries are missing the income field.

- If we train a model on this incomplete data, it might misclassify applicants:
- Someone with a high income might be flagged as risky.
- Someone with low income might be approved incorrectly.

Cleaning the data by filling missing income values or removing incomplete records improves the model's predictions.

## Summary

- Good data preparation ensures accuracy, reliability, and fairness in machine learning.
- Skipping this step can lead to wrong decisions and lost trust in your models.

# Data Science Life Cycle Context

- Collect → Prepare → Analyze → Evaluate → Deploy
- For today: **Prepare** = Preprocessing + Cleaning

## 1.Preprocessing

- Transforming the data into a format that a machine learning model can understand.

Example: Converting text labels (“Yes”/“No”) into numbers (1/0), scaling values, or encoding categories.

## 2.Cleaning (subset of preprocessing)

- Fixing problems in the data, such as missing values, duplicates, or errors.

Example: If a dataset has missing income values, we might fill them in or remove those rows.

# Definition of Data Cleaning

- Data cleaning is the process of finding and fixing problems in your dataset so it is correct and usable.
- Dirty or messy data can confuse machine learning models, leading to wrong results.

## Tasks include:

**Missing values** – some data points are empty or not reported.

**Outliers** – values that are unusually high or low compared to the rest.

**Duplicates** – repeated rows or records in the dataset.

**Wrong formats** – inconsistent ways of writing dates, numbers, or categories.

# Why Cleaning is Important

- **Accuracy:** Clean data ensures the model predicts correctly.
- **Efficiency:** Models train faster when there's no noise or errors.
- **Trust:** Stakeholders and decision-makers can rely on the results.
- **Ethics:** Reduces bias and error, and prevents unfair decisions, especially in finance (e.g., loan approvals).
- Saves time and resources in later analysis.

# Common Errors in Financial Data

**Duplicates:** Same transaction recorded more than once → could inflate totals.

**Inconsistencies:** Different ways of writing currency (USD, \$, US\$) → causes confusion in calculations.

**Missing values:** Some fields like income or credit history might be empty → affects predictions.

**Wrong formats:** Dates written differently (“2025/18/09” vs “2025-09-18”) → model may fail to read them.

# Real Finance Example

**Dataset:** Loan applications from a bank.

**Problems:**

- 30% of applicants missing income information.
- Some entries show negative age values (impossible).

**Solution:**

**Clean the data** – detect missing and wrong entries.

**Impute missing values** – fill in missing income with an average or estimated value.

**Remove invalid entries** – delete impossible values like negative age.

**Result:** The dataset becomes reliable for predicting loan approvals.

# Consequences of Poor Cleaning

**Wrong risk predictions:** Bank might approve high-risk applicants or reject safe ones.

Example: A bank's loan dataset has missing income data. A model predicts a high-income applicant as low-income due to missing values. The bank wrongly rejects a reliable borrower or approves someone who might default.

**Fraud detection failures:** Duplicate or inconsistent data can hide suspicious activity.

Example: A dataset has duplicate transactions from a credit card. Fraud detection software ignores them as normal activity. A fraudster can charge the same amount multiple times without being flagged.

**Incorrect investment signals:** Poor data may mislead analysts into making bad financial decisions.

Example: A stock trading dataset records prices in different formats (USD, \$). An algorithm misreads them, seeing a stock price as \$1000 instead of \$10. Analysts get false signals and may make bad investment decisions, like buying or selling at the wrong time.

# Retrieving & Inspecting Data

## Sources of Financial Data

- CSV and JSON files – e.g., stock prices, transactions.
- Databases – e.g., customer bank records.
- APIs – e.g., stock market API, financial services.
- Cloud sources – scalable and secure storage.

# Python setup

For lab activity, we will be using the following libraries:

- **pandas** for managing the data.
- **numpy** for mathematical operations.
- **seaborn** for visualizing the data.
- **matplotlib** for visualizing the data.
- **sklearn** for machine learning and machine-learning-pipeline related functions.
- **scipy** for statistical computations.

# Demo Lab1: Reading CSV in Python

```
import pandas as pd
```

```
# Reading a CSV file
```

```
loans = pd.read_csv("loan_data.csv")
```

```
print(loans.head())
```

```
# Reading a JSON file
```

```
transactions = pd.read_json("transactions.json")
```

```
print(transactions.head())
```

# Demo Lab2: Reading CSV in Python

```
import pandas as pd
```

```
# Example: Loan dataset
```

```
data = {  
    "age": [25, 40, None, 35, -5],  
    "salary": [50000, 60000, 55000, None, 45000],  
    "loan_amount": [20000, 25000, 22000, 18000, 30000],  
    "default": ["No", "Yes", "No", "No", "Yes"]  
}
```

```
loans = pd.DataFrame(data)  
print(loans)
```

Understanding Data Issues  
From the example dataset,  
notice:

- **Missing Values:** age (row 3), salary (row 4).
- **Outlier/Wrong Value:** age = -5 (not realistic).
- **Target Variable:** default (Yes/No).

**Output Table:**

| age | salary | loan_amount | default |
|-----|--------|-------------|---------|
| 25  | 50000  | 20000       | No      |
| 40  | 60000  | 25000       | Yes     |
| NaN | 55000  | 22000       | No      |
| 35  | NaN    | 18000       | No      |
| -5  | 45000  | 30000       | Yes     |

- These issues must be fixed before analysis or ML modeling.

# Reading & understanding our dataset : (Inspecting the Dataset)

# Basic info

loans.info()

# Summary statistics

loans.describe()

- info() → column types, non-null counts
- describe() → mean, std, min, max, quartiles
  - Helps identify missing values and suspicious entries (e.g., negative age)

# Reading & understanding our dataset : (Checking for Duplicates and Unique Index)

# Check duplicates

```
print(loans.duplicated())
```

# Drop duplicates if any

```
loans = loans.drop_duplicates()
```

# Check if index is unique

```
print(loans.index.is_unique)
```

- `drop_duplicates()`: Ensures dataset is clean and unique before ML
- Important for financial datasets to avoid double counting loans

# Demo Lab: Checking Missing Values

```
# Check for missing values  
print(loans.isnull().sum())
```

## Output:

age → 1 missing

salary → 1 missing

**isnull():** Checks each value in the DataFrame to see if it is missing (i.e., NaN in Python).

Output: True where the value is missing and False where it is not.

**sum():** When applied to a DataFrame of boolean values, sum() treats True as 1 and False as 0, and counts how many missing values there are in each column.

# Sorting and Inspecting Extreme Values

# Count missing values per column

```
print(loans.isnull().sum().sort_values())
```

# Top 2 largest loans

```
print(loans.sort_values(by='loan_amount', ascending=False)[:2])
```

# Top 2 highest salaries

```
print(loans.sort_values(by='salary', ascending=False)[:2])
```

`sort_values()`: Shows missing values sorted from least to most.

- Helps prioritize cleaning steps (age & salary)
- Useful to identify extreme values or outliers

# Demo Lab: Handling Missing Values

# Option 1: Drop missing values

```
loans_drop = loans.dropna()
```

# Option 2: Fill with mean

```
loans_fill = loans.copy()
```

```
loans_fill["age"].fillna(loans_fill["age"].mean(), inplace=True)
```

```
loans_fill["salary"].fillna(loans_fill["salary"].mean(), inplace=True)
```

**dropna():** Removes rows (or columns) that have missing values (NaN). A smaller dataset, but no missing values.

**fillna():** Replaces missing values with a specified value (like mean, median, 0, or a custom value).

❖ In finance: Missing salary can be replaced with average salary of customers.

# Demo Lab: Handling Missing Values

#Filling Missing Values: Categorical

# If default had missing values, fill with mode

```
loans["default"].fillna(loans["default"].mode()[0], inplace=True)
```

- `mode()` → finds the most frequent value (mode) in the categorical column.
- `mean()` → calculates the average value of a column.

# Demo Lab: Handling Wrong / Outlier Values

```
# Detect outliers (z-score method)
```

```
z_scores=np.abs(stats.zscore(loans_fill[["age","salary","loan_amount"]]))
```

```
# Replace negative ages with median age
```

```
loans_clean = loans_fill.copy()
```

```
loans_clean.loc[loans_clean["age"] < 0, "age"] = loans_fill["age"].median()
```

**Z-scores()**: Measures how far a value is from the mean in terms of standard deviations.

Large absolute z-scores (commonly  $> 3$ ) indicate outliers (negative age  $\rightarrow$  high z-score).

**Median**: The middle value of a dataset when sorted.

Example: For [10, 20, 30], median = 20

Less affected by outliers than the mean.

# Demo Lab: Handling Outlier (Visualizing Outliers)

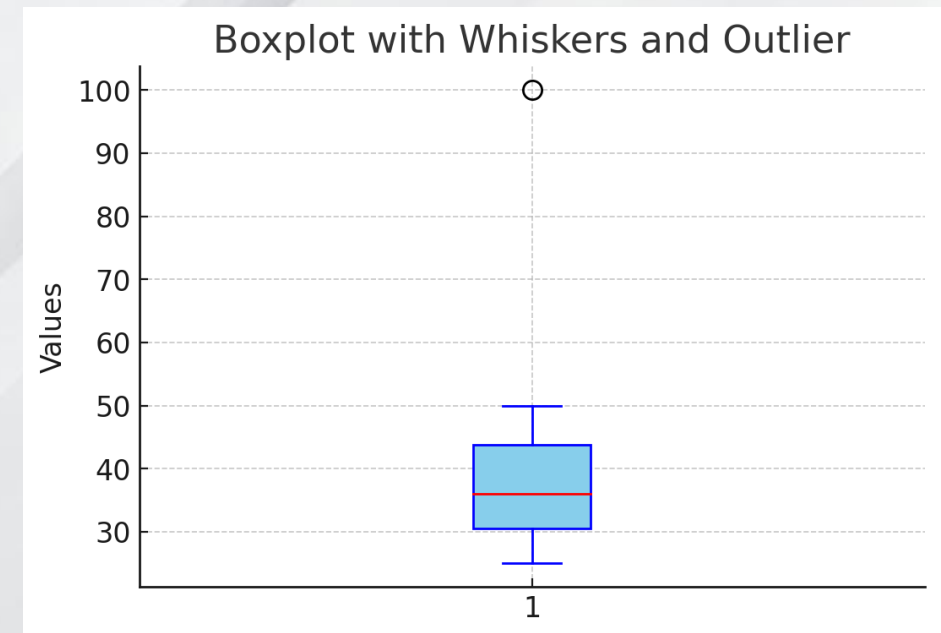
```
!pip install seaborn
import seaborn as sns
import matplotlib.pyplot as plt

# Boxplot for numeric columns (Creates a boxplot for age to visualize its spread
and outliers)

sns.boxplot(x=loans["age"])
plt.title("Age Distribution & Outliers")
plt.show()
```

**Boxplot**: A graphical way to see the distribution of numeric data.

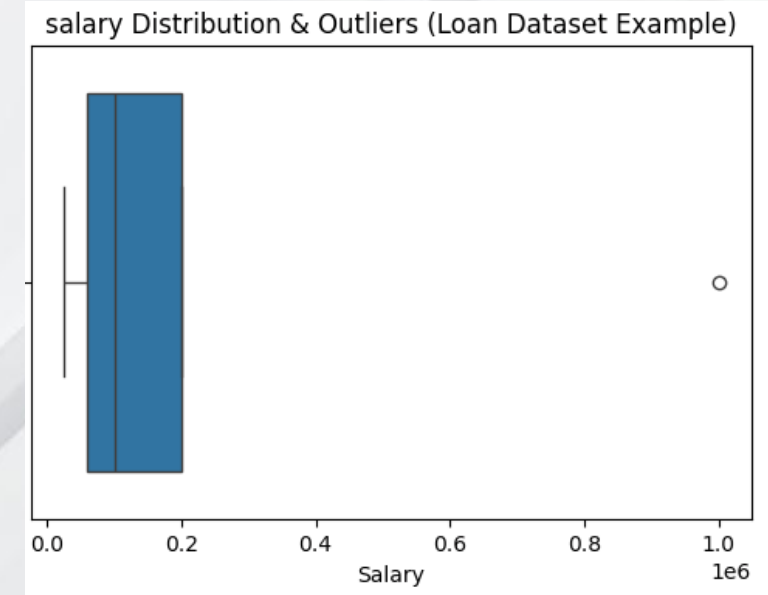
Shows: Median (middle line), Quartiles (box edges), Outliers (points outside the whiskers)



# Demo Lab: Handling Outlier (Visualizing Outliers)

```
sns.boxplot(x=loans["salary"])  
plt.title("Salary Distribution & Outliers")
```

- Boxplots make it easy to see the center, spread, and unusual values in numeric columns.
- `x=column` in `sns.boxplot()`: Specifies which column of your DataFrame to plot on the x-axis.
- Plots the age column along the x-axis.
- If you use `y=column`, it would plot the values vertically instead.
- In short: `x=` tells Seaborn which data to visualize horizontally.



# Normalizing Data

```
from sklearn.preprocessing import MinMaxScaler
```

```
scaler = MinMaxScaler()
```

```
loans[["age","salary","loan_amount"]] =  
scaler.fit_transform(loans[["age","salary","loan_amount"]])
```

```
print(loans)
```

MinMaxScaler():

- Scales numeric features between 0–1
- Essential for ML algorithms (SVM, KNN)

Dataset after normalization:

|   | age      | salary   | loan_amount | default |
|---|----------|----------|-------------|---------|
| 0 | 0.076923 | 0.357143 | 0.166667    | No      |
| 1 | 1.000000 | 1.000000 | 0.583333    | Yes     |
| 2 | 0.000000 | 0.714286 | 0.333333    | No      |
| 3 | 0.692308 | 0.535714 | 0.000000    | No      |
| 4 | 0.076923 | 0.000000 | 1.000000    | Yes     |

# Lab Activity: Data preparation 1

```
# =====  
# Lecture 2 Demo: Data Preparation I  
# =====  
  
# Approved libraries  
import pandas as pd  
import numpy as np  
import seaborn as sns  
import matplotlib.pyplot as plt  
from sklearn.preprocessing import MinMaxScaler  
from scipy import stats  
%matplotlib inline
```

# Lab Activity: Data preparation 1

```
# -----
```

```
# 1. Create loan dataset
```

```
# -----
```

```
data = {  
    "age": [25, 40, None, 35, -5],  
    "salary": [50000, 60000, 55000, None, 45000],  
    "loan_amount": [20000, 25000, 22000, 18000, 30000],  
    "default": ["No", "Yes", "No", "No", "Yes"]  
}
```

```
loans = pd.DataFrame(data)  
print("Original Dataset:")  
display(loans)
```

# Lab Activity: Data preparation 1

```
# -----
```

```
# 2. Inspecting the dataset
```

```
# -----
```

```
print("\n--- Info ---") print(loans.info())
```

```
print("\n--- Describe ---") print(loans.describe().round(3))
```

```
print("\n--- Check for duplicates ---") print(loans.duplicated())
```

```
loans = loans.drop_duplicates()
```

```
print("After dropping duplicates:")
```

```
display(loans)
```

```
print("\n--- Check if index is unique ---") print(loans.index.is_unique)
```

# Lab Activity: Data preparation 1

```
# -----
```

```
# 3. Checking missing values
```

```
# -----
```

```
print("\n--- Missing Values ---")
```

```
print(loans.isnull().sum().sort_values())
```

```
# Visualize missing data
```

```
sns.heatmap(loans.isnull(), cbar=False, cmap="viridis")
```

```
plt.title("Missing Values in Loan Dataset")
```

```
plt.show()
```

# Lab Activity: Data preparation 1

```
# -----
```

```
# 4. Inspecting extreme values
```

```
# -----
```

```
print("\n--- Top 2 largest loan amounts ---")
```

```
print(loans.sort_values(by='loan_amount', ascending=False)[:2])
```

```
print("\n--- Top 2 highest salaries ---")
```

```
print(loans.sort_values(by='salary', ascending=False)[:2])
```

# Lab Activity: Data preparation 1

```
# -----
```

```
# 5. Handling missing values
```

```
# -----
```

```
# Numeric columns
```

```
loans["age"].fillna(loans["age"].mean(), inplace=True)
```

```
loans["salary"].fillna(loans["salary"].median(), inplace=True)
```

```
# Categorical column (if any missing, not in this case)
```

```
loans["default"].fillna(loans["default"].mode()[0], inplace=True)
```

```
print("\nDataset after filling missing values:")
```

```
display(loans)
```

# Lab Activity: Data preparation 1

```
# -----
```

```
# 6. Detecting outliers (Z-score method)
```

```
# -----
```

```
numeric_cols = ["age", "salary", "loan_amount"]
```

```
z_scores = np.abs(stats.zscore(loans[numeric_cols]))
```

```
print("\n--- Z-scores ---")
```

```
display(pd.DataFrame(z_scores, columns=numeric_cols))
```

# Lab Activity: Data preparation 1

```
# -----  
# 7. Treating outliers  
# -----  
# Replace negative ages with median  
median_age = loans["age"].median()  
loans.loc[loans["age"] < 0, "age"] = median_age  
  
# Cap salary outliers (95th percentile)  
upper_salary = loans["salary"].quantile(0.95)  
loans["salary"] = np.where(loans["salary"] > upper_salary, upper_salary, loans["salary"])  
  
print("\nDataset after treating outliers:")  
display(loans)
```

# Lab Activity: Data preparation 1

```
# -----
```

```
# 8. Visualizing cleaned data
```

```
# -----
```

```
# Boxplots for numeric columns
```

```
for col in numeric_cols:
```

```
    sns.boxplot(x=loans[col])
```

```
    plt.title(f"{col} Distribution & Outliers")
```

```
    plt.show()
```

# Lab Activity: Data preparation 1

```
# Pairplot for relationships
```

```
sns.pairplot(loans, hue="default")
```

```
plt.show()
```

```
# -----
```

```
# 9. Normalizing numeric columns
```

```
# -----
```

```
scaler = MinMaxScaler()
```

```
loans[numeric_cols] = scaler.fit_transform(loans[numeric_cols])
```

```
print("\nDataset after normalization:")
```

```
display(loans)
```

## Example of Demo Output

- Dataset after filling missing values
- Filled missing values with mean (numeric) and mode (categorical).

Dataset after filling missing values:

|   | age   | salary  | loan_amount | default |
|---|-------|---------|-------------|---------|
| 0 | 25.00 | 50000.0 | 20000       | No      |
| 1 | 40.00 | 60000.0 | 25000       | Yes     |
| 2 | 23.75 | 55000.0 | 22000       | No      |
| 3 | 35.00 | 52500.0 | 18000       | No      |
| 4 | -5.00 | 45000.0 | 30000       | Yes     |

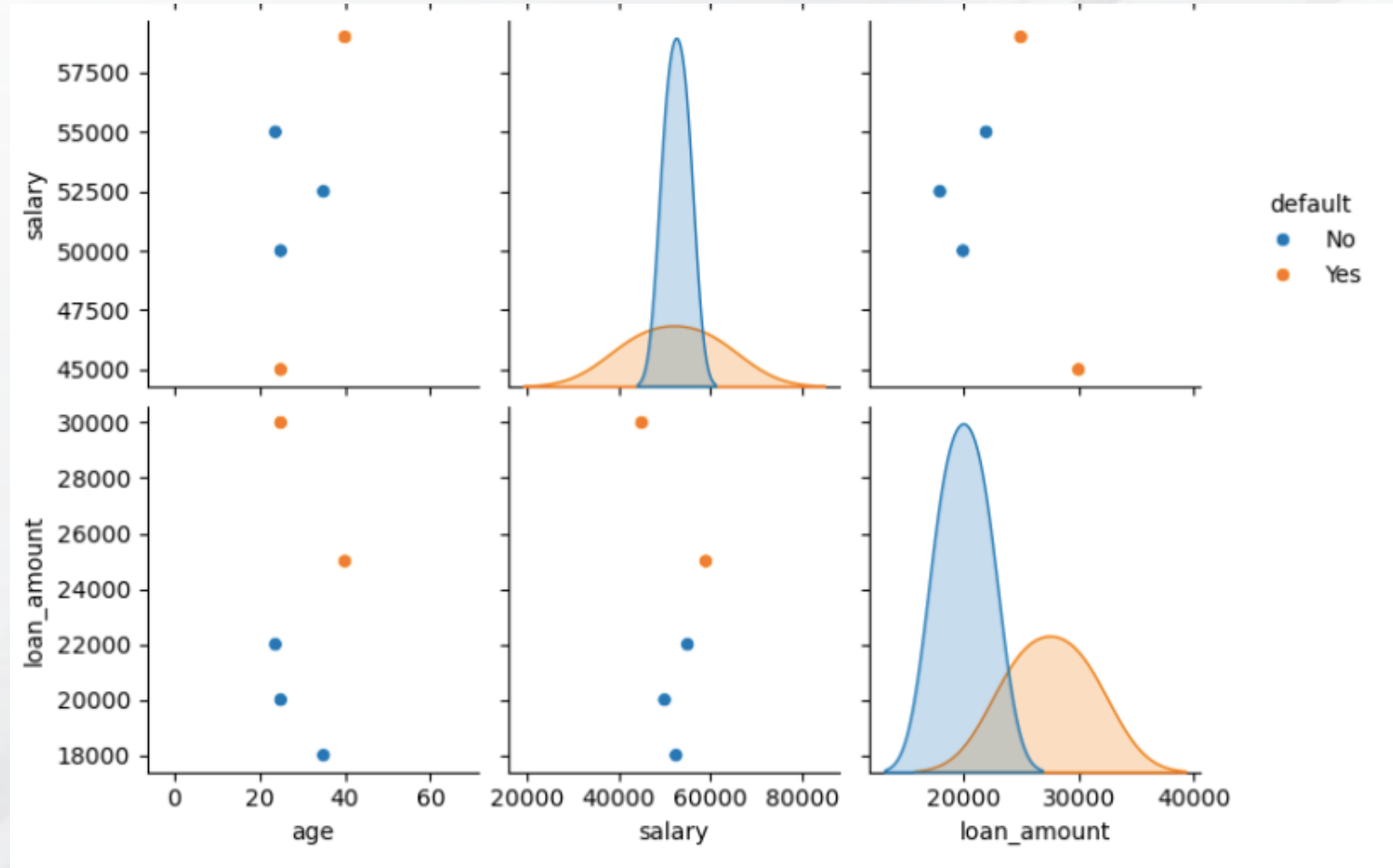
## Example of Demo Output

- This heatmap provides a visual overview of missing values in the loan dataset, making data cleaning decisions easier.
- Helps decide whether to drop or fill missing values.



# Example of Demo Output

- Orange color = people who defaulted.
- Blue color = people who did not default.
- Look at each scatterplot → do the colors mix together or do they separate?
- If they separate → those variables help predict default.
- If they mix → no strong relationship.



# What this demo does?

- Creates a loan dataset with missing and wrong values.
- Uses `info()`, `describe()`, `duplicated()`, `index.is_unique`, `isnull().sum().sort_values()` for inspection.
- Visualizes missing values with a heatmap.
- Handles missing numeric and categorical values.
- Detects outliers with z-scores.
- Treats outliers (negative age → median, salary → capped at 95th percentile).
- Visualizes the cleaned dataset with boxplots and pairplots.
- Normalizes numeric columns with `MinMaxScaler` for ML readiness.

# Recap: Data Preparation I

- Data cleaning ensures accuracy, consistency, and reliability of financial datasets.
- We covered how to:
- Retrieve data from CSV, JSON, SQL databases, APIs, and cloud sources.
- Inspect datasets with `info()`, `describe()`, `duplicated()`, `isnull().sum()`.
- Handle missing values using mean, median, mode, and categorical imputation.
- Detect and treat outliers (negative values, z-score, capping).
- Remove or merge duplicates.
- Apply sorting and indexing for better data quality control.
- Normalize features with `MinMaxScaler` for ML readiness.
- Cleaned data is the foundation for trustworthy machine learning models in finance.

# References

- Han, J., Kamber, M., & Pei, J. (2011). Data Mining: Concepts and Techniques. Morgan Kaufmann.
- Géron, A. (2019). Hands-On Machine Learning with Scikit-Learn, Keras, and TensorFlow. O'Reilly Media.
- Provost, F., & Fawcett, T. (2013). Data Science for Business. O'Reilly Media.
- Kuhn, M., & Johnson, K. (2013). Applied Predictive Modeling. Springer.
- McKinney, W. (2017). Python for Data Analysis: Data Wrangling with Pandas, NumPy, and IPython. O'Reilly.
- VanderPlas, J. (2016). Python Data Science Handbook. O'Reilly.
- Articles from IEEE Xplore and Springer on data preprocessing in financial machine learning.

A circular graphic composed of various blue icons related to machine learning and artificial intelligence. The icons include a brain with circuitry, a gear, a robot, a computer monitor, a speech bubble, a head with circuitry, a brain, a gear, a robot, a computer monitor, a speech bubble, a head with circuitry, and a brain. The text "MACHINE LEARNING" is written in a bold, blue, sans-serif font across the center of the graphic.

# MACHINE LEARNING

Thank you!