UNIVERSITÄT LEIPZIG
Medizinische Fakultät

Universitätsklinikum Leipzig
Medizin ist unsere Berufung.

# Data Handling and Preprocessing for Tabular Clinical Data

Data Science and AI for Medicine - Training School 2026

MDS
Medical Data Science

# Why Preprocessing Matters

- **Raw data** from hospitals, laboratories, or electronic health records (EHRs) often contains:
  - **Errors** – typos (e.g. 17.1m instead of 1.71m)
  - **Inconsistencies** - same variable stored differently (e.g., “male”, “M”, “m”)
  - **Missing values** - follow-up information not recorded
  - **Outliers** - extremely high or low lab values

→ Problems distort research results and bias AI models

→ Clean data = more reliable analyses and safer medical decision-making

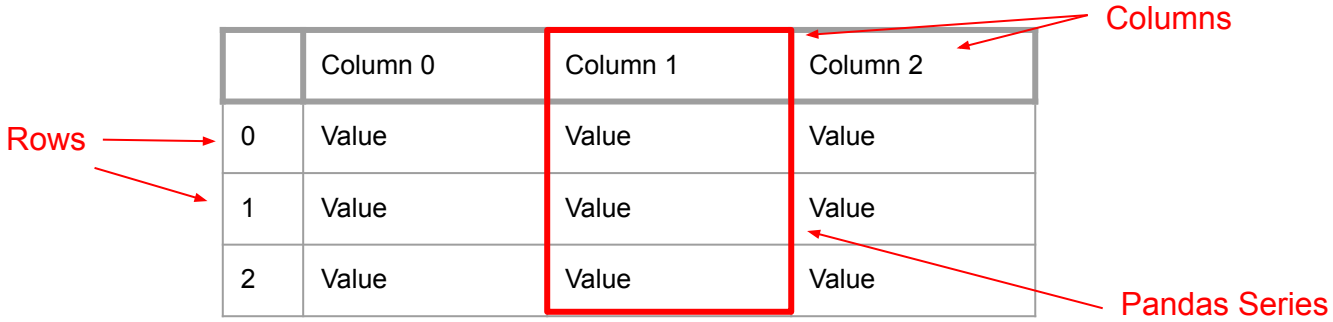
https://www.pexels.com/de-de/foto/marketing-angebot-mann-hande-5816299/

# What is pandas?

- pandas is a Python library for data handling and analysis
  - https://pandas.pydata.org/
- It works especially well with tabular data (similar to Excel tables)
- Built on NumPy, with seamless integration into the Python data ecosystem
- pandas is the de facto standard for working with tabular data in Python

# Core *pandas* Structures

- Series:
  - One-dimensional labeled array (like a single column)
- DataFrame:
  - Two-dimensional table with rows and columns
  - Each row = one patient record
  - Each column = one variable (e.g., patient ID, gender, admission date)
- pandas makes it easy to filter, merge, and summarize datasets




| | Column 0 | Column 1 | Column 2 |
|---|---|---|---|
| 0 | Value | Value | Value |
| 1 | Value | Value | Value |
| 2 | Value | Value | Value |

# Exploring DataFrames

- Always start with a data overview:
  - `df.head()` → first few rows
  - `df.info()` → datatypes, missing values, number of rows
  - `df.describe()` → summary stats (mean, mi max, outliers)
- To check completeness, detect errors early, understand dataset scope

```
# Summary statistics
df.describe(include="all")
```
✓ [9] 85ms

1-10 | 11 rows × 15 cols

| | Name | Age | Gender | Blood Type |
|---|---|---|---|---|
| count | 55500 | 55500.000000 | 55500 | 55500 |
| unique | 49992 | NaN | 2 | 8 |
| top | DAvId muNoZ | NaN | Male | A- |
| freq | 3 | NaN | 27774 | 6969 |
| mean | NaN | 51.539459 | NaN | NaN |
| std | NaN | 19.602454 | NaN | NaN |
| min | NaN | 13.000000 | NaN | NaN |
| 25% | NaN | 35.000000 | NaN | NaN |
| 50% | NaN | 52.000000 | NaN | NaN |
| 75% | NaN | 68.000000 | NaN | NaN |

```
# Data types and missing values
df.info()
```
✓ [8] 24ms

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 55500 entries, 0 to 55499
Data columns (total 15 columns):
 #   Column              Non-Null Count  Dtype
---  ------              --------------  -----
 0   Name                55500 non-null  object
 1   Age                 55500 non-null  int64
 2   Gender              55500 non-null  object
 3   Blood Type          55500 non-null  object
```

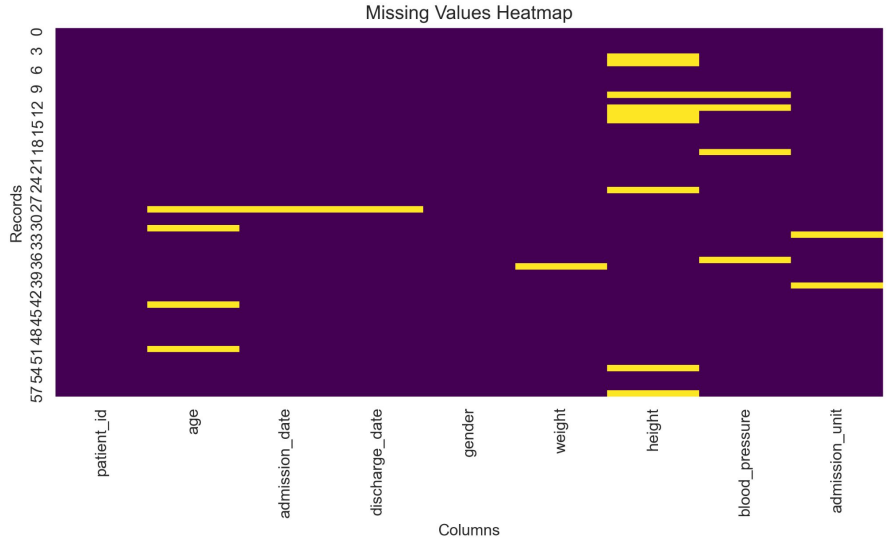
```
df.head()
```
✓ [5] 134ms

5 rows | 5 rows × 15 cols

| | Name | Age | Gender | Blood Type |
|---|---|---|---|---|
| 0 | Bobby JacksOn | 30 | Male | B- |
| 1 | LesLie TErRy | 62 | Male | A+ |
| 2 | DaNnY sMitH | 76 | Female | A- |
| 3 | andrEw waTtS | 28 | Female | O+ |

# Visual Exploration

- Before and after cleaning, always explore your data visually
  - **Histograms** show distribution of numeric variables
  - **Boxplots** reveal outliers
  - **Missing value heatmaps** highlight gaps
- Visualizations make patterns and errors obvious, validate preprocessing decisions and detect mistakes



# Step 1 - Data Cleaning: Handling Data Types

- Wrong datatypes lead to wrong results (e.g., dates or numbers stored as strings)
- Check datatypes:
  - `df.dtypes´

```
1  # check dtypes
2  df.dtypes
   ✓ [31] < 10 ms
```

9 rows  9 rows × 1 cols

| | <unnamed> |
|---|---|
| patient_id | int64 |
| age | float64 |
| admission_date | object |
| discharge_date | object |
| gender | object |
| weight | float64 |

- Convert types:
  - Dates: `df['admission_date'] = pd.to_datetime(df['admission_date'])´
  - Numbers: `pd.to_numeric(df['age'], errors='coerce')´
- Medical example:
  - Length of stay = discharge date – admission date → only works if dates are datetime

# Handling Missing Data

- Why it matters in clinical data:
  - Missingness is extremely common (e.g., missing labs, unrecorded weight)
  - Ignoring missing data can bias results

- How to detect missing values:
  - `df.isnull().sum()` → count missing values per column

```
1  # Count missing values per column
2  df.isnull().sum()
```

✓ [10] 30ms

1-10 15 rows × 1 cols

| | 123 <unnamed> |
|---|---|
| Name | 0 |
| Age | 0 |
| Gender | 0 |
| Blood Type | 0 |

# Handling Missing Data

- How to handle them
  - Remove incomplete rows

    ```
    df.dropna(subset=['admission_date'])
    ```
  - Impute missing values:
    - Median age for missing age values:

      ```
      df['age'].fillna(df['age'].median(), inplace=True)
      ```
    - Forward-fill for longitudinal time series data:

      ```
      df.fillna(method='ffill')
      ```

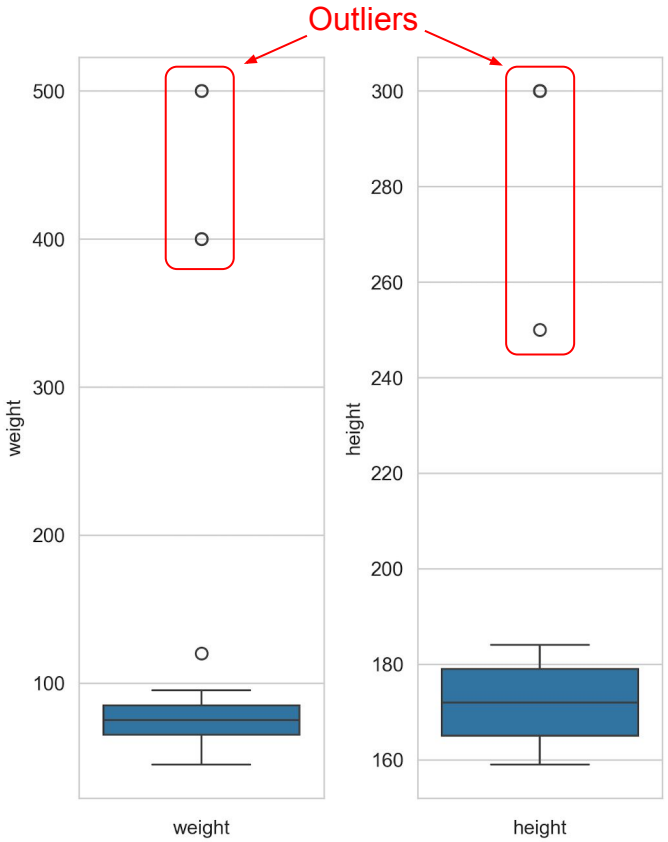
➔ **Strategy depends on clinical context (don’t blindly fill!)**

# Types of Missing Data

- **MCAR** (Missing Completely At Random):
  - No relationship to other data
  - **Example**: a lab machine randomly fails for some patients
- **MAR** (Missing At Random):
  - Related to other variables
  - **Example**: cholesterol missing more often in younger patients
- **MNAR** (Missing Not At Random):
  - Depends on the unobserved value itself
  - **Example**: patients with very high BMI avoid reporting weight
- Visualization helps to detect patterns of missingness help distinguish between MCAR, MAR, and MNAR

# Handling Outliers

- Outliers may represent errors (e.g., weight = 500 kg), or real but rare events (e.g., extremely high BP in ICU)
- Detect with pandas:
  - `df['weight'].describe()` → min, max, quartiles
- Visualization: `sns.boxplot(x=df['weight'])`
- Possible strategies:
  - Exclude implausible values (clear errors)
  - Cap extreme but possible values
  - Keep rare but clinically valid cases (flag them for review)



# Handling Duplicates

- Why it matters in medicine:
  - Duplicate patient records can lead to double-counting
  - Example: a patient transferred between units may appear multiple times
- Detect duplicates:
  - `df.duplicated().sum()` → count duplicates
- Remove duplicates:
  - `df.drop_duplicates(inplace=True)`
- Decision needed: sometimes duplicates are real (e.g., re-admissions) → check clinical logic

```
1  # Check for duplicates
2  df.duplicated().sum()
✓ [12] 49ms
```

```
np.int64(534)
```

```
1  # Remove duplicate rows
2  df = df.drop_duplicates()
✓ [13] 51ms
```

# Step 2 - Data Integration – Merging Data Sources

- Why it matters:
  - Clinical data often comes from **multiple systems** (labs, ICU, billing)
  - Must be merged for meaningful analysis
- pandas merge:
  - `df.merge(labs, on='patient_id', how='left')`
- Join types:
  - Inner: only patients with labs
  - Left: keep all patients, add lab values if available
- **Challenges**:
  - Different schemas (column names, formats)
  - Patient matching (IDs, pseudonyms)
  - Conflicts in values

# Step 3 - Data Transformation - Feature Engineering

- What it means: transform raw data into features that have (clinical) meaning
- Examples:
  - New features: `df['BMI'] = df['weight'] / (df['height']/100)**2`
  - Group and summarize: `df.groupby('admission_unit')['age'].mean()`
  - Pivot tables: `df.pivot_table(index='admission_unit', values='albumin_g_dl', aggfunc='mean')`
- Why it matters:
  - Provides variables clinicians understand (BMI instead of raw height & weight)
  - Can improve model accuracy by encoding medical knowledge
  - Enhances interpretability for clinical decision-making

# Encoding Categorical Data

- ML Algorithms require numbers, not text
- Check categories:
  - `df['gender'].unique()`
- Encoding:
  - One-hot encoding: `pd.get_dummies(df['admission_unit'])`
  - Binary mapping: `df['gender'].map({'male':0, 'female':1})`
  - Label encoding: `LabelEncoder().fit_transform(df['column_name'])`
- Medical examples:
  - Admission units: ICU, ER, Surgery → one-hot encoded for analysis

```
1  # Map Gender to numeric
2  df["Gender_num"] = (df["Gender"]
3                      .map({"Male": 0,
4                            "Female": 1}))
```

| 123 Gender_num |
|---|
| 0 |
| 0 |
| 1 |
| 1 |
| 1 |

| Admission Type_Elective | Admission Type_Emergency | Admission Type_Urgent |
|---|---|---|
| False | False | True |
| False | True | False |
| False | True | False |
| True | False | False |
| False | False | True |

# Scaling & Encoding

- Why it matters:
  - Lab values, vital signs, and scores are measured in different units
  - Scaling ensures comparability across features (e.g., blood pressure vs. albumin)
- Min-Max scaling → rescales values to [0, 1]

```
from sklearn.preprocessing import MinMaxScaler
    df['bp_scaled']=MinMaxScaler().fit_transform(df[['blood_pressure']])
```

- Standardization (Z-score) → mean = 0, std = 1

```
from sklearn.preprocessing import StandardScaler
    df['age_scaled'] = StandardScaler().fit_transform(df[['age']])
```

# Step 4 - Data Reduction - Filtering & Subsetting Data

- **Reducing features**
  - High-dimensional data → harder to analyze, risk of overfitting
  - Reduction strategies:
    - Drop irrelevant variables
    - Correlation analysis (drop highly correlated features)
    - Dimensionality reduction (PCA, t-SNE, UMAP)
- **Subsetting**
  - Focus on relevant patient groups for analysis
  - Examples:
    - Patients with diabetes who attended follow-up
    - Patients above 65 years admitted in winter months
  - Enables targeted, clinically relevant analyses

# Key Takeaways

- **Preprocessing** is critical for reliable analysis and safe use of AI in medicine
- **Visualization** is essential to detect hidden problems
- **Missing data** handling must consider the clinical context
- **Outliers** may be errors or clinically meaningful extremes — treat carefully
- **Feature engineering, scaling,** and **encoding** prepare data for valid analysis
- Always **document** your decisions to ensure **reproducibility** and **trustworthiness**