Data Cleaning

Folder Structure

```
.
├── data/
│   └── MIMIC-ED/
│       ├── event_level_v2.csv
│       ├── event_level_cleaned_v2.csv
│       ├── event_level_cleaned_sirs_v2.csv
│       ├── event_level_training_data.csv
│       ├── cox_static_random_train.csv
│       ├── cox_timevarying_train.csv
│       ├── cox_static_landmark_stacked_train.csv
│       └── discrete_time_30min_train.csv
│
├── data_cleaning/
│   ├── nan_finding.ipynb
│   ├── nan_remover.py
│   ├── data_cleaner.py
│   ├── lab_adder.py
│   ├── sirs_adder.py
│   ├── survival_analysis_reformatter.py
│   ├── train_data_maker.ipynb
│   ├── discrete_survival_reformatter.py
│   └── MIMIC_DATA_CLEANING.py
│
└── README.md
```

1. NaN Analysis (`nan_finding.ipynb`)

Purpose

- Identify missingness patterns in the raw event-level data
- Validate improvements after NaN cleaning
- Support QA before downstream preprocessing

Pipeline

1. Load `event_level_v2.csv`
2. Compute NaN counts per column
3. Load `event_level_cleaned_v2.csv`
4. Compare pre- and post-cleaning counts
5. Output a summary report

Outputs

- Printed NaN summary report inside the notebook

2. NaN Removal (`nan_remover.py`)

Purpose

- Apply forward-fill and backward-fill strategies per stay
- Replace remaining missingness with global medians
- Produce a cleaned event-level dataset

Pipeline

1. Group by `stay_id`
2. Sort rows by timestamp
3. Forward-fill numeric columns
4. Backward-fill remaining NaNs
5. Replace residual NaNs using global medians

Outputs

- `event_level_cleaned_v2.csv`

3. General Event-Level Cleaner (`data_cleaner.py`)

Purpose

- Standardize column types, IDs, and timestamp formats
- Remove malformed or duplicate rows
- Prepare the dataset for lab merging and SIRS computation

Pipeline

1. Normalize column names
2. Convert timestamps
3. Validate numeric datatypes

4. Save cleaned intermediate file

Outputs

- `event_level_cleaned_base.csv` (intermediate)

4. Laboratory Value Merger (`lab_adder.py`)

Purpose

- Add lactate and WBC laboratory measurements
- Standardize column names
- Ensure consistent timestamp alignment

Pipeline

1. Load raw lab event files
2. Map labs to unified naming conventions
3. Merge labs into event-level dataset
4. Save merged dataset

Outputs

- `event_level_with_lac_wbc.csv`

5. SIRS & Sepsis Onset Labeling (`sirs_adder.py`)

Purpose

- Merge WBC using a 24-hour backward as-of join
- Compute SIRS components (Temp, HR, RR, WBC)
- Identify earliest SIRS≥2 or antibiotic administration
- Generate leakage-safe sepsis onset labels

Pipeline

1. Parse timestamps and coerce IDs
2. Perform `merge_asof` with WBC file
3. Compute SIRS component flags
4. Identify SIRS and antibiotic timestamps
5. Define sepsis onset per stay
6. Add `is_sepsis_onset` and `is_sepsis`

Outputs

- `event_level_cleaned_sirs_v2.csv`

6. Initial Survival Formatting (`survival_analysis_reformatter.py`)

Purpose

- Prepare dataset for survival modeling
- Remove leakage-prone columns
- Convert timestamps into numerical durations

Pipeline

1. Load SIRS-augmented dataset
2. Drop label-related columns
3. Convert timestamps to datetime
4. Compute `time_since_adm`

Outputs

- `event_level_survival_ready.csv`

7. Training Data Maker — Classification Dataset (`train_data_maker.ipynb`)

Purpose

- Construct a balanced septic vs. non-septic dataset
- Engineer features for ML models

Pipeline

1. Load `event_level_with_lac_wbc.csv`
2. Drop high-missingness columns
3. Select septic stays
4. Sample matching non-septic stays
5. Compute `time_since_adm`
6. One-hot encode top medication GSN codes
7. Remove leakage columns

Outputs

- `event_level_training_data.csv`

8. Training Data Maker — Survival Datasets (`train_data_maker.ipynb`)

Purpose

- Generate survival-compatible datasets for multiple frameworks

Pipeline

1. Compute event and censor times
2. Build:
    - Static Cox snapshot dataset
    - Time-varying Cox interval dataset
    - Landmark-stacked survival dataset
3. Encode variables
4. Drop leak-prone features

Outputs

- `cox_static_random_train.csv`
- `cox_timevarying_train.csv`
- `cox_static_landmark_stacked_train.csv`

9. Discrete-Time Survival Reformatter (`discrete_survival_reformatter.py`)

Purpose

- Convert irregular event-level measurements into a discrete-time person-period format
- Support logistic hazard models and neural survival models

Pipeline

1. Load survival-ready dataset
2. Compute remaining time to event for each landmark row
3. Generate 30-minute discrete time bins
4. Assign event labels to correct bin

5. Output person-period table

Outputs

- `discrete_time_30min_train.csv`

10. Full Pipeline Wrapper (`MIMIC_DATA_CLEANING.py`)

Purpose

- Optional script to run the entire cleaning and preprocessing pipeline end-to-end

Pipeline

1. Execute NaN removal
2. Perform lab merging
3. Apply SIRS logic
4. Run survival formatting
5. Build classification and survival datasets

Outputs

- All processed datasets written to `data/MIMIC-ED/`

Running the Pipeline

Prerequisites

- Python 3.10+
- Install dependencies

Run

1. python data_cleaning/nan_finding.ipynb
2. python data_cleaning/nan_remover.py
3. python data_cleaning/data_cleaner.py
4. python data_cleaning/lab_adder.py
5. ppython data_cleaning/sirs_adder.py
6. python data_cleaning/train_data_maker.ipynb
7. python data_cleaning/survival_analysis_reformatter.py
8. python data_cleaning/discrete_survival_reformatter.py

9. python data_cleaning/MIMIC_DATA_CLEANING.py