02_Base_Data_Construction

June 8, 2025

1 Base Data Construction

1.1 Environment Setup and Utility Function for Exporting DataFrames

To ensure a clean and modular development workflow, this initial code block sets up the necessary environment paths and imports the foundational Python libraries required for data manipulation. The variable `PATH` points to the raw data directory where original MIMIC-III files reside, while `EXPORT_PATH` designates a location for storing intermediate or processed data artifacts, following best practices in reproducible data science workflows.

The function `export_to_csv()` encapsulates the logic for exporting `pandas` DataFrames to CSV files. Before saving, it performs a check to verify whether the target directory exists, creating it if necessary. This simple yet effective safeguard ensures that any downstream function can persist data without manual folder creation, thus supporting automation and modular execution of the entire data pipeline.

This type of setup, while elementary, is essential for enabling scalable and organized experimentation, especially in the context of complex datasets such as MIMIC-III, where intermediate artifacts are frequently generated during data cleaning, integration, and feature engineering.

[ ]:
```python
EXPORT_PATH = "../data/processed/"
PATH = "../data/raw/"

import os
import pandas as pd
import numpy as np

def export_to_csv(df, filename):
    """
    Exports a DataFrame to a CSV file.

    Parameters:
    df (pd.DataFrame): The DataFrame to export.
    filename (str): The name of the file to save the DataFrame to.
    """
    if not os.path.exists(EXPORT_PATH):
        os.makedirs(EXPORT_PATH)
    df.to_csv(os.path.join(EXPORT_PATH, filename), index=False)
```

1.1.1 Cohort Initialization: Loading Filtered Sepsis Patients

The current step initializes the working cohort by loading a pre-filtered set of patient records diagnosed with sepsis, as stored in the file `sepsis_cohort.csv`. This cohort is assumed to have been previously constructed based on the presence of specific ICD9 codes associated with sepsis, as outlined in `D_ICD_DIAGNOSES.csv`. By retaining only the `SUBJECT_ID` and `HADM_ID` columns and applying `drop_duplicates()`, the script ensures that each patient–admission pair is represented uniquely. This is critical to avoid redundancy in downstream joins and to ensure consistency when aggregating clinical events. The final line, `cohort['SUBJECT_ID'].nunique()`, provides a quick check on the number of **unique patients** included in the study. This count serves both as a sanity check and as a summary statistic to track cohort size across preprocessing steps.

[ ]:
```
# Load filtered sepsis cohort (previously generated)
sepsis_ids = pd.read_csv(os.path.join(EXPORT_PATH, 'sepsis_cohort.csv'))
cohort = sepsis_ids[['SUBJECT_ID', 'HADM_ID']].drop_duplicates()
display(cohort.head())
cohort['SUBJECT_ID'].nunique()
```

	SUBJECT_ID	HADM_ID
0	51797	104616
1	44534	183659
2	14828	144708
3	14828	125239
4	44500	101872

[ ]: 3068

1.1.2 ICU Stay Filtering for Sepsis Cohort

To refine the initial patient-level sepsis cohort into a set of valid ICU admissions, this block performs a multi-step filtering process on the `ICUSTAYS` table from MIMIC-III. The goal is to retain only **clinically meaningful ICU stays**, ensuring that each row corresponds to a valid episode of intensive care associated with a confirmed sepsis diagnosis.

The inner join between `icustays` and the previously constructed `cohort` ensures that only ICU stays related to previously filtered sepsis admissions are retained. Further cleaning is applied through several exclusion criteria:

- `LOS.notnull()` and `LOS > 0`: Removes entries with undefined or non-positive length of stay, which are often artifacts or incomplete discharges.
- `OUTTIME > INTIME`: Ensures logical temporal consistency of the ICU stay, excluding corrupted or incomplete entries.
- `drop_duplicates(subset=["ICUSTAY_ID"])`: Retains only unique ICU stay identifiers, preventing redundancy when a patient is admitted to the ICU multiple times during the same hospital stay.

This refined DataFrame, `cohort_icu`, now constitutes the **core analytical population** for all subsequent data aggregation, visualization, and modeling efforts.

[ ]:
```
icustays = pd.read_csv(os.path.join(PATH, 'icustays.csv'))

cohort_icu = icustays.merge(cohort, on=["SUBJECT_ID", "HADM_ID"], how="inner")
cohort_icu = cohort_icu[cohort_icu["LOS"].notnull() & (cohort_icu["LOS"] > 0)]
cohort_icu = cohort_icu[cohort_icu["OUTTIME"] > cohort_icu["INTIME"]]
cohort_icu = cohort_icu.drop_duplicates(subset=["ICUSTAY_ID"])

print(f"Valid ICU Admissions for Cohort: {cohort_icu.shape[0]}")
display(cohort_icu[["SUBJECT_ID", "HADM_ID", "ICUSTAY_ID", "LOS"]].head())
```

```
Valid ICU Admissions for Cohort: 3685
```

	SUBJECT_ID	HADM_ID	ICUSTAY_ID	LOS
0	269	106296	206613	3.2788
1	275	129886	219649	7.1314
2	292	179726	222505	0.8854
3	305	194340	217232	2.4370
4	323	143334	264375	3.0252

1.1.3 Demographic and Admission Enrichment of ICU Cohort

This section performs a critical enrichment of the ICU-based cohort by integrating **demographic** and **admission-level variables**, which are essential for understanding patient profiles and risk factors. The first merge integrates data from the `PATIENTS` table, bringing in `GENDER` and `DOB` for each `SUBJECT_ID`. This enables computation of patient **age at admission**, which is a known confounding variable in ICU outcomes and is often correlated with mortality, severity of illness, and resource consumption. The second merge pulls from the `ADMISSIONS` table, adding context-specific variables such as:

- `ADMITTIME`: Timestamp of hospital admission.
- `ADMISSION_TYPE`: Scheduled vs emergency nature of the admission.
- `INSURANCE` and `ADMISSION_LOCATION`: Socioeconomic and referral indicators.
- `HOSPITAL_EXPIRE_FLAG`: A critical binary outcome reflecting in-hospital mortality, useful for cohort stratification.

Age Computation: Age is calculated by subtracting the birth year from the admission year, with a further adjustment to correct for patients whose birthdays fall later in the calendar year. Finally, the `AGE` variable is capped at 91 to comply with **privacy masking procedures** used in MIMIC-III, where patients older than 89 are anonymized.

This enriched DataFrame `df` now contains a mix of temporal, demographic, and categorical indicators, providing a well-rounded feature space for exploratory analysis and machine learning.

[ ]:
```
patients = pd.read_csv(os.path.join(PATH, 'PATIENTS.csv'), parse_dates=["DOB"])
admissions = pd.read_csv(os.path.join(PATH, 'ADMISSIONS.csv'),␣
 ↪parse_dates=["ADMITTIME"])

df = cohort_icu.merge(patients[["SUBJECT_ID", "GENDER", "DOB"]],␣
 ↪on="SUBJECT_ID", how="left")
```

```
df = df.merge(admissions[[
    "SUBJECT_ID", "HADM_ID", "ADMITTIME", "ADMISSION_TYPE",␣
 ↪"ADMISSION_LOCATION",
    "INSURANCE", "HOSPITAL_EXPIRE_FLAG"
]], on=["SUBJECT_ID", "HADM_ID"], how="left")

df["AGE"] = df["ADMITTIME"].dt.year - df["DOB"].dt.year
adjust = ((df["ADMITTIME"].dt.month < df["DOB"].dt.month) |
          ((df["ADMITTIME"].dt.month == df["DOB"].dt.month) &
           (df["ADMITTIME"].dt.day < df["DOB"].dt.day)))
df["AGE"] -= adjust.astype(int)
df["AGE"] = df["AGE"].clip(upper=91)
```

Temporal Feature Extraction: Hour of Day and Day of Week In this stage, the dataset is further enriched with **time-derived features** from existing timestamp columns, specifically `INTIME` (ICU admission time) and `ADMITTIME` (hospital admission time). These datetime fields are parsed into standard `pandas` datetime objects using `pd.to_datetime()` with error coercion to ensure robustness against malformed entries.

Two temporal descriptors are extracted for each timestamp:

- `*_HOUR`: The hour of the day, ranging from 0 to 23, capturing circadian patterns which may be associated with shifts in hospital staffing, admission policies, or physiological rhythms in patients.
- `*_WEEKDAY`: The day of the week, encoded from 0 (Monday) to 6 (Sunday), which allows for the analysis of potential variations in care delivery, ICU availability, or admission frequency across the week.

These engineered features, though simple, can uncover hidden periodicities in patient outcomes or ICU operational behavior, and are especially relevant for models where interpretability and feature salience are evaluated.

[ ]:
```
timestamp_cols = ["INTIME", "ADMITTIME"]
for col in timestamp_cols:
    df[col] = pd.to_datetime(df[col], errors="coerce")
    df[f"{col}_HOUR"] = df[col].dt.hour
    df[f"{col}_WEEKDAY"] = df[col].dt.weekday

# Weekend effect
df['WEEKEND_ADMISSION'] = (df['ADMITTIME_WEEKDAY'] >= 5).astype(int)
df['WEEKEND_ICU'] = (df['INTIME_WEEKDAY'] >= 5).astype(int)

# Night shift effect
df['NIGHT_ADMISSION'] = ((df['ADMITTIME_HOUR'] >= 19) |
                         (df['ADMITTIME_HOUR'] < 7)).astype(int)

# Time between hospital and ICU admission
```

```
df['ADMIT_TO_ICU_HOURS'] = (df['INTIME'] - df['ADMITTIME']).dt.total_seconds() /
 ↪ 3600

# Seasonal patterns
df['SEASON'] = df['ADMITTIME'].dt.month % 12 // 3 + 1
df['QUARTER'] = df['ADMITTIME'].dt.quarter
```

1.1.4 Final Static Dataset Construction and Export

In this final step of the data preparation pipeline, a curated set of variables is selected to construct the foundational dataset `df_final`. This dataset consists solely of **static or quasi-static features**, i.e., variables that are either fixed at the time of ICU admission or derived from static tables such as `PATIENTS` and `ADMISSIONS`.

The selected features include:

- **Identifiers**: `SUBJECT_ID`, `HADM_ID`, `ICUSTAY_ID`
- **Demographics**: `AGE`, `GENDER`
- **Admission Characteristics**: `ADMISSION_TYPE`, `ADMISSION_LOCATION`, `INSURANCE`, `FIRST_CAREUNIT`
- **Clinical Outcomes**: `LOS` (Length of Stay), `HOSPITAL_EXPIRE_FLAG`
- **Temporal Signatures**: `*_HOUR` and `*_WEEKDAY` from both `INTIME` and `ADMITTIME`

Before exporting the final DataFrame, a null check on the `LOS` variable ensures that only rows with a valid target value are retained, reinforcing the integrity of downstream supervised learning tasks.

Finally, the dataset is serialized to a CSV file under the `processed/` directory for persistence and modularity. This checkpoint marks the end of the static data integration pipeline, yielding a well-structured dataset for visualization and predictive modeling.

[ ]:
```
# df_final = df[[
#     "SUBJECT_ID", "HADM_ID", "ICUSTAY_ID", "AGE", "GENDER",
#     "ADMISSION_TYPE", "ADMISSION_LOCATION", "INSURANCE",
#     "FIRST_CAREUNIT", "LOS", "HOSPITAL_EXPIRE_FLAG",
#     "INTIME_HOUR", "INTIME_WEEKDAY", "ADMITTIME_HOUR", "ADMITTIME_WEEKDAY",␣
 ↪"INTIME"
# ]]

# df_final = df_final[df_final["LOS"].notnull()]
print(f"df_final shape: {df.shape}")

# Define columns of interest for clarity and modularity
final_cols = [
    "SUBJECT_ID", "HADM_ID", "ICUSTAY_ID", "AGE", "GENDER",
    "ADMISSION_TYPE", "ADMISSION_LOCATION", "INSURANCE",
    "FIRST_CAREUNIT", "LOS", "HOSPITAL_EXPIRE_FLAG",
    "INTIME_HOUR", "INTIME_WEEKDAY", "ADMITTIME_HOUR", "ADMITTIME_WEEKDAY",␣
 ↪"INTIME"
]
```

```
df_final = df[final_cols].dropna(subset=["LOS"])
print(f"df_final shape: {df.shape}")
df_final.to_csv(EXPORT_PATH + "df_final_static.csv", index=False)
display(df_final.head())
```

```
df_final shape: (3685, 24)
df_final shape: (3685, 24)
```

	SUBJECT_ID	HADM_ID	ICUSTAY_ID	AGE	GENDER	ADMISSION_TYPE	ADMISSION_LOCATION	INSURANCE	FIRST_CAREUNIT	LOS	HOSPITAL_EXPIRE_FLAG	INTIME_HOUR	INTIME_WEEKDAY	ADMITTIME_HOUR	ADMITTIME_WEEKDAY	INTIME
0	269	106296	206613	40	M	EMERGENCY	EMERGENCY ROOM ADMIT	Medicaid	MICU	3.2788	0	11	0	11	0	2170-11-05 11:05:29
1	275	129886	219649	82	M	EMERGENCY	EMERGENCY ROOM ADMIT	Medicare	CCU	7.1314	1	11	6	3	5	2170-10-07 11:28:53
2	292	179726	222505	57	F	URGENT	TRANSFER FROM HOSP/EXTRAM	Private	MICU	0.8854	1	18	3	18	3	2103-09-27 18:29:30
3	305	194340	217232	76	F	EMERGENCY	TRANSFER FROM HOSP/EXTRAM	Medicare	SICU	2.4370	1	12	5	18	5	2129-09-03 12:31:31
4	323	143334	264375	57	M	EMERGENCY	EMERGENCY ROOM ADMIT	Medicare	MICU	3.0252	0	15	3	15	3	2120-01-11 15:48:28

[ ]:
```
# Install needed packages
!apt-get install texlive texlive-xetex texlive-latex-extra pandoc &> /dev/null
!pip install pypandoc &> /dev/null

# Mount your google drive to get access to your ipynb files

from google.colab import drive
drive.mount('/content/drive')
# and copy your notebook to this colab machine. Note that I am using *MY*␣
 ↪notebook filename
```

```
!cp "/content/drive/MyDrive/Colab Notebooks/02_Base_Data_Construction.ipynb" ./␣
 ↪&> /dev/null

# Then you can run the converter.

!jupyter nbconvert --to PDF "02_Base_Data_Construction.ipynb" &> /dev/null
```