

# FindHAPISFinal

November 29, 2025

- 0.1 This code identifies Hospital-Acquired Pressure Injuries (HAPIs) in the MIMIC-IV dataset by combining structured ICD-9/ICD-10 pressure-ulcer codes with unstructured clinical note analysis and applying the 48-hour hospital-acquired timing rule.

```
[12]: #Import Libraries
import pandas as pd #For Data Manipulation
from datetime import timedelta #For Time Manipulation
import re #For Regular Expressions

#Sets max rows and columns to display when printing dataframes
pd.set_option("display.max_rows", 30)
pd.set_option("display.max_columns", 20)
```

```
[13]: # File Paths
ADMISSIONS_PATH = r"D:\School\5141\admissions.csv\admissions.csv"
DISCHARGE_PATH = r"D:\School\5141\discharge.csv\discharge.csv"
DIAGNOSES_PATH = r"D:\School\5141\diagnoses_icd.csv\diagnoses_icd.csv"

# Output Path
OUTPUT_PATH = r"D:\School\5141\hospitalwide_hapi_labels.csv"
```

```
[14]: # List of ICD codes for pressure ulcers
# ICD-10 codes from https://www.cms.gov/icd10m/version35-fullcode-cms/
# fullcode_cms/P0227.html
icd10_pu_codes = [
    "L89000", "L89001", "L89002", "L89003", "L89004", "L89009",
    "L89010", "L89011", "L89012", "L89013", "L89014", "L89019",
    "L89020", "L89021", "L89022", "L89023", "L89024", "L89029",
    "L89100", "L89101", "L89102", "L89103", "L89104", "L89109",
    "L89110", "L89111", "L89112", "L89113", "L89114", "L89119",
    "L89120", "L89121", "L89122", "L89123", "L89124", "L89129",
    "L89130", "L89131", "L89132", "L89133", "L89134", "L89139",
    "L89140", "L89141", "L89142", "L89143", "L89144", "L89149",
    "L89150", "L89151", "L89152", "L89153", "L89154", "L89159",
    "L89200", "L89201", "L89202", "L89203", "L89204", "L89209",
    "L89210", "L89211", "L89212", "L89213", "L89214", "L89219",
    "L89220", "L89221", "L89222", "L89223", "L89224", "L89229",
```

```

    "L89300", "L89301", "L89302", "L89303", "L89304", "L89309",
    "L89310", "L89311", "L89312", "L89313", "L89314", "L89319",
    "L89320", "L89321", "L89322", "L89323", "L89324", "L89329",
    "L8940", "L8941", "L8942", "L8943", "L8944", "L8945",
    "L89500", "L89501", "L89502", "L89503", "L89504", "L89509",
    "L89510", "L89511", "L89512", "L89513", "L89514", "L89519",
    "L89520", "L89521", "L89522", "L89523", "L89524", "L89529",
    "L89600", "L89601", "L89602", "L89603", "L89604", "L89609",
    "L89610", "L89611", "L89612", "L89613", "L89614", "L89619",
    "L89620", "L89621", "L89622", "L89623", "L89624", "L89629",
    "L89810", "L89811", "L89812", "L89813", "L89814", "L89819",
    "L89890", "L89891", "L89892", "L89893", "L89894", "L89899",
    "L8990", "L8991", "L8992", "L8993", "L8994", "L8995"
]
# ICD-9 codes from https://qualityindicators.ahrq.gov/Downloads/Modules/PSI/V50/
# TechSpecs/PSI_03_Pressure_Ulcer_Rate.pdf
icd9_pu_codes = [
    "7070", "70700", "70701", "70702", "70703", "70704",
    "70705", "70706", "70707", "70709",
    "70720", "70721", "70722", "70723", "70724", "70725"
]

# Combine ICD-9 and ICD-10 codes into a set
PRESSURE_ULCER_CODES = set([c.strip().upper() for c in (icd10_pu_codes +
    icd9_pu_codes)])

```

[15]: # Create a list of wound/pressure ulcer keywords to search for in clinical notes

```

WOUND_KEYWORDS = [
    "ulcer", "pressure ulcer", "pressure injury",
    "decubitus", "bedsore", "skin breakdown",
    "non-blanch", "nonblanch", "non blanch",
    "erythema", "unstageable", "deep tissue", "dti", "sdtpi",
    "stage 1", "stage i", "stage 2", "stage ii", "stage 3", "stage iii",
    "stage 4", "stage iv",
    "sacral wound", "sacral ulcer", "coccyx ulcer",
    "heel ulcer", "ischial ulcer",
]
# Wound keywords into a regex pattern
"""
Lowercase both the pattern and the note text, so matching is effectively
case-insensitive.
re.escape ensures any special characters in keywords are treated literally.
"""

WOUND_PATTERN = re.compile("|".join([re.escape(k.lower()) for k in
    WOUND_KEYWORDS]))

```

```
[16]: # Load admissions table (hadm_id & admittime)
"""
Only need the hospital admission ID (hadm_id) and the admission timestamp
↳(admittime)
so we can calculate how many hours after admission the first wound-related note
↳appears.
This is necessary for identifying potential HAPIs using the 48-hour rule.
"""

admissions = pd.read_csv(
    ADMISSIONS_PATH,
    usecols=["hadm_id", "admittime"],
    low_memory=False # Prevents dtype-guessing issues when reading large CSVs
)

# Ensure correct data types for merging and time calculations
admissions["hadm_id"] = admissions["hadm_id"].astype("Int64")
admissions["admittime"] = pd.to_datetime(admissions["admittime"])

print("Admissions shape:", admissions.shape)
```

Admissions shape: (546028, 2)

```
[17]: # Process discharge notes in chunks to identify wound-related notes
"""
The discharge notes file is very large, so it is read in chunks
to avoid memory issues. For each chunk, we normalize text, search for
wound-related keywords using a compiled regex pattern, and keep only the
notes that match. All wound-related notes are collected into a single list.
"""

chunksize = 100_000 #Number of rows per chunk when reading the CSV
wound_notes = []

reader = pd.read_csv(
    DISCHARGE_PATH,
    usecols=["hadm_id", "charttime", "text"],
    chunksize=chunksize,
    low_memory=False
)

# Iterate chunk-by-chunk over the discharge notes
for i, chunk in enumerate(reader, start=1):

    # Drop rows where hadm_id is missing
    chunk = chunk.dropna(subset=["hadm_id"])

    # Normalize types and text format
    chunk["hadm_id"] = chunk["hadm_id"].astype("Int64")
```

```

chunk["text"] = chunk["text"].astype(str).str.lower()

# Detect wound-related notes using the compiled regex pattern
chunk["is_wound"] = chunk["text"].apply( # is_wound = True if any keyword
    ↪is found in the note
        lambda t: bool(WOUND_PATTERN.search(t))
    )

# Keep only notes flagged as wound-related
wound_chunk = chunk[chunk["is_wound"]].copy()

# Convert charttime to datetime and store results in a list titled
↪wound_notes
if not wound_chunk.empty:
    wound_chunk["charttime"] = pd.to_datetime(
        wound_chunk["charttime"], errors="coerce"
    )
    wound_notes.append(wound_chunk)

# Combine all wound-related notes into a single DataFrame
if wound_notes:
    all_wound_notes = pd.concat(wound_notes, ignore_index=True) # DataFrame of
    ↪all wound-related notes
else:
    all_wound_notes = pd.
    ↪DataFrame(columns=["hadm_id", "charttime", "text", "is_wound"]) # Empty
    ↪DataFrame with correct columns

print("Total wound notes:", len(all_wound_notes))

```

Total wound notes: 195795

[18]: # Identify the earliest wound-related note time per admission

```

"""
From the full set of wound-related notes, we keep the earliest (first) wound
documentation for each hos hadm_id. This timestamp will be
used to calculate how many hours after admission the first wound note appears.
"""

if not all_wound_notes.empty:
    first_wound_note = (
        all_wound_notes
            .dropna(subset=["charttime"]) # Ensure charttime is present
            .sort_values(["hadm_id", "charttime"]) # Sort so the earliest note
    ↪comes first
            .groupby("hadm_id", as_index=False) # Group by admission
            .first()[["hadm_id", "charttime"]] # Take the first wound note per
    ↪hadm_id

```

```

        .rename(columns={"charttime": "first_wound_note_time"}) # Rename for clarity
    )
else:
    # If no wound-related notes were found, create an empty table with expected columns
    first_wound_note = pd.DataFrame(
        columns=["hadm_id", "first_wound_note_time"]
    )

```

```
[19]: # Merge admissions with first wound note & apply 48-hour rule

"""
Merge the earliest wound-note time with the admissions table so we can calculate the time difference between admission and the first wound-related documentation. We then apply a 48-hour threshold to approximate HAPIs.
"""

labels = admissions.merge(first_wound_note, on="hadm_id", how="left")

# Calculate time (in hours) from admission to first wound-related note
labels["hours_after_admit"] = (
    labels["first_wound_note_time"] - labels["admittime"]
).dt.total_seconds() / 3600.0

# notes_pu = 1 if the admission has any wound-related note at all
labels["notes_pu"] = labels["first_wound_note_time"].notna().astype(int)

# Initialize unstructured HAPI label
HAPI_THRESHOLD = 48 # Threshold in hours for HAPI definition
labels["HAPI_UNSTRUCTURED"] = 0

# Valid rows where we could compute hours_after_admit
mask_valid = labels["hours_after_admit"].notna()

# Set HAPI_UNSTRUCTURED = 1 if first wound note occurs more than 48 hours after admission
labels.loc[
    (labels["hours_after_admit"] > HAPI_THRESHOLD) & mask_valid,
    "HAPI_UNSTRUCTURED"
] = 1

# If the wound note appears before admission (negative hours), treat as non-hospital-acquired
labels.loc[
    (labels["hours_after_admit"] < 0) & mask_valid,
    "HAPI_UNSTRUCTURED"
] = 0
```

```
print("HAPI_UNSTRUCTURED distribution:")
print(labels["HAPI_UNSTRUCTURED"].value_counts())
```

```
HAPI_UNSTRUCTURED distribution:
HAPI_UNSTRUCTURED
0    412105
1    133923
Name: count, dtype: int64
```

```
[20]: # 7. STRUCTURED LABEL (ICD-based pressure ulcer codes)
```

```
"""
Use the diagnoses_icd table to create a structured HAPI label.
Any hadm_id with at least one pressure-ulcer ICD-9/ICD-10 code in
↳PRESSURE_ULCER_CODES is flagged as HAPI_STRUCTURED = 1.

"""

dx = pd.read_csv(DIAGNOSES_PATH, low_memory=False)

# Normalize ICD codes and ensure hadm_id type matches other tables
dx["icd_code"] = dx["icd_code"].astype(str).str.upper().str.strip()
dx["hadm_id"] = dx["hadm_id"].astype("Int64")

# Keep only rows with pressure-ulcer ICD codes
pu_dx = dx[dx["icd_code"].isin(PRESSURE_ULCER_CODES)]

# Collapse to one row per admission with at least one PU code
pu_by_hadm = (
    pu_dx.groupby("hadm_id")
    .size()
    .reset_index(name="pu_count")      # pu_count = number of PU diagnoses for
    ↳that admission
)
pu_by_hadm["HAPI_STRUCTURED"] = 1      # Any PU code → structured HAPI flag = 1
pu_by_hadm = pu_by_hadm[["hadm_id", "HAPI_STRUCTURED"]]

# Merge structured label onto the existing labels DataFrame
labels = labels.merge(pu_by_hadm, on="hadm_id", how="left")

# Admissions with no matching PU codes get HAPI_STRUCTURED = 0
labels["HAPI_STRUCTURED"] = labels["HAPI_STRUCTURED"].fillna(0).astype(int)

print("HAPI_STRUCTURED distribution:")
print(labels["HAPI_STRUCTURED"].value_counts())
```

```
HAPI_STRUCTURED distribution:
HAPI_STRUCTURED
0    537574
```

```
1      8454  
Name: count, dtype: int64
```

```
[21]: # Combine structured & unstructured evidence  
"""  
Define a final HAPI label (HAPI_FINAL) that incorporates both structured  
(ICD-based) and unstructured (notes & 48-hour rule) data. An admission  
is considered a HAPI case if either source indicates a HAPI.  
"""  
labels["HAPI_FINAL"] = (  
    (labels["HAPI_STRUCTURED"] == 1) |  
    (labels["HAPI_UNSTRUCTURED"] == 1)  
).astype(int)  
  
# Print final label distribution  
print("HAPI final distribution:")  
print(labels["HAPI_FINAL"].value_counts())
```

```
HAPI final distribution:  
HAPI_FINAL  
0      408955  
1      137073  
Name: count, dtype: int64
```

```
[22]: # Save final label table  
"""  
Save the final label table, including timing information, notes-based flags,  
structured ICD flags, and the combined HAPI_FINAL label. This file will be  
used downstream for feature engineering and model training.  
"""  
final_labels = labels[[  
    "hadm_id",  
    "admittime",  
    "first_wound_note_time",  
    "hours_after_admit",  
    "notes_pu",  
    "HAPI_UNSTRUCTURED",  
    "HAPI_STRUCTURED",  
    "HAPI_FINAL",  
]  
  
final_labels.to_csv(OUTPUT_PATH, index=False) #Save to output path  
print("Saved to:", OUTPUT_PATH) # Check to make sure path is correct
```

```
Saved to: D:\School\5141\hospitalwide_hapi_labels.csv
```