U.PORTO

University of Porto

Machine Learning Project

Predicting ICU Length of Stay Using Machine Learning: A Comparative Study on MIMIC-III

Authors:
Giuseppe Pitruzzella
Radvilė Rušaitė
Karlota Bochanaitė

Professor:
Inês Dutra

00_Introduction

June 8, 2025

1 Supervised Modeling Pipeline for ICU Length of Stay Prediction

Authors: *Giuseppe Pitruzzella, Radvilė Rušaitė, Karlota Bochanaitė*

This project is situated within the field of supervised learning, aiming to predict a continuous clinical variable—**Length of Stay (LOS)** in the Intensive Care Unit—using regression models based on neural networks. The approach aligns with fundamental machine learning paradigms studied during the course, including probabilistic optimization techniques such as Maximum A Posteriori (MAP) estimation.

The analysis uses real-world data from **MIMIC-III** (Medical Information Mart for Intensive Care), a publicly available clinical database developed by the MIT Lab for Computational Physiology in collaboration with Beth Israel Deaconess Medical Center (Boston). The database contains de-identified health-related data associated with over 60,000 ICU admissions between 2001 and 2012, and has become a globally recognized benchmark for research in medical data science and critical care.

Dataset Setup To facilitate reproducibility, all necessary .csv files from the MIMIC-III clinical dataset have been kindly provided by the course instructor in compressed format (.csv.gz). The files have been made available via an educational mirror for use in this project.

Environment Setup Before proceeding with the analysis, make sure all required Python libraries are available. You can automatically install missing dependencies listed in `requirements.txt` by executing the following cell.

[ ]:
```
import subprocess
import sys
import importlib.util

# Path to the requirements.txt file
req_file = "../requirements.txt"  # modify if necessary

def read_requirements(file_path):
    with open(file_path, "r") as f:
        return [line.strip().split("==")[0] for line in f if line.strip() and␣
 ↪not line.startswith("#")]

def is_installed(package_name):
```

```
    return importlib.util.find_spec(package_name) is not None

def pip_install(package_name):
    print(f"Installing: {package_name}")
    subprocess.check_call([sys.executable, "-m", "pip", "install",␣
 ↪package_name])

# Map exceptions between package name and importable module
module_map = {
    "scikit-learn": "sklearn",
    "ipython": "IPython"
}

# Load packages from requirements.txt
required_packages = read_requirements(req_file)

# Install only missing packages
for pkg in required_packages:
    module_name = module_map.get(pkg, pkg)
    if not is_installed(module_name):
        pip_install(pkg)
    else:
        print(f"Already installed: {pkg}")
```

```
Already installed: torch
Already installed: pandas
Already installed: matplotlib
Already installed: seaborn
Already installed: numpy
Already installed: scikit-learn
Already installed: xgboost
Already installed: ipython
```

1.1 Initial Exploration

We begin by exploring the main tables in the MIMIC-III dataset. The goal of this first phase is to understand the structure of the database and identify relevant variables that may influence ICU length of stay. Key reference tables include:

- `D_ICD_DIAGNOSES.csv` – diagnosis codes (ICD9)
- `ICUSTAYS.csv` – ICU stays metadata
- `D_ITEMS.csv` – item IDs and descriptions for time-series events

[ ]:
```
# 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/00_Introduction.ipynb" ./ &> /dev/
 ↪null

# Then you can run the converter.

!jupyter nbconvert --to PDF "00_Introduction.ipynb" &> /dev/null
```

01_Data_Exploration

June 8, 2025

0.1 Dataset Understanding

The first phase of any data-centric project, particularly in the healthcare domain, requires a thorough understanding of the structure, semantics, and scope of the data sources involved. This notebook is dedicated to the initial exploration of the MIMIC-III clinical database, a publicly available dataset that includes de-identified health-related data associated with over 60,000 intensive care unit (ICU) admissions. The database encompasses a wide range of structured tables capturing demographic information, administrative details, diagnoses, laboratory results, vital signs, prescriptions, and more. The primary goal of this chapter is to provide a systematic overview of the core MIMIC-III tables that are fundamental to our modeling task. By performing an initial inspection of each dataset—examining their dimensionality, column names, data types, and representative records—we establish a foundational understanding that will inform downstream preprocessing, cohort definition, and feature engineering strategies.

To preserve computational feasibility during initial exploration, only a sample of rows is loaded for the largest tables. Full data ingestion will be deferred to the feature engineering phase, where filters based on cohort definitions and ICU stay windows will be applied. This initial exploration enables a critical appraisal of data completeness, granularity, and linkage keys across tables, and lays the groundwork for selecting an appropriate disease cohort and engineering predictive variables for ICU length of stay modeling.

[ ]:
```
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
from IPython.display import display
```

[ ]:
```
PATH='../data/raw/'
EXPORT_PATH = '../data/processed/'
ASSETS_PATH='../assets/plots/data_exploration/'

patients = pd.read_csv(PATH + "PATIENTS.csv")
admissions = pd.read_csv(PATH + "ADMISSIONS.csv")
icustays = pd.read_csv(PATH + "ICUSTAYS.csv")
diagnoses = pd.read_csv(PATH + "DIAGNOSES_ICD.csv")
d_icd_diagnoses = pd.read_csv(PATH + "D_ICD_DIAGNOSES.csv")
chartevents = pd.read_csv(PATH + "CHARTEVENTS.csv", nrows=100)
labevents = pd.read_csv(PATH + "LABEVENTS.csv", nrows=5000)
inputevents_mv = pd.read_csv(PATH + "INPUTEVENTS_MV.csv", nrows=5000)
inputevents_cv = pd.read_csv(PATH + "INPUTEVENTS_CV.csv", nrows=5000)
```

```
outputevents = pd.read_csv(PATH + "OUTPUTEVENTS.csv", nrows=5000)
prescriptions = pd.read_csv(PATH + 'PRESCRIPTIONS.csv', usecols=['HADM_ID',␣
↪'DRUG'])
```

0.2 Demographic and Mortality Data of Patients

PATIENTS.csv provides demographic and survival information for each patient included in the MIMIC-III database. This dataset is essential for identifying individual patients and for understanding general demographic patterns, such as gender distribution, age-related trends, and mortality rates. The file allows the integration of demographic information with clinical and physiological data, enabling a comprehensive assessment of patient characteristics and long-term outcomes after critical care.

[ ]:
```
print(patients.shape)
display(patients.head())
```

```
(46520, 8)
```

	ROW_ID	SUBJECT_ID	GENDER	DOB	DOD	\
0	234	249	F	2075-03-13 00:00:00	NaN	
1	235	250	F	2164-12-27 00:00:00	2188-11-22 00:00:00	
2	236	251	M	2090-03-15 00:00:00	NaN	
3	237	252	M	2078-03-06 00:00:00	NaN	
4	238	253	F	2089-11-26 00:00:00	NaN	

	DOD_HOSP	DOD_SSN	EXPIRE_FLAG
0	NaN	NaN	0
1	2188-11-22 00:00:00	NaN	1
2	NaN	NaN	0
3	NaN	NaN	0
4	NaN	NaN	0

0.3 Hospital Admission History and Patient Care Pathways

ADMISSIONS.csv documents each hospitalization event of patients registered in the MIMIC-III database. This dataset is critical for tracking the history of care, analyzing admission patterns, and studying the causes of hospitalization and clinical outcomes of critically ill patients. The table provides a comprehensive view of the patient pathway within the health care facility, allowing analyses on the impact of length of stay, frequency of hospitalizations, and key diagnostics associated with admission. It also allows linking different clinical episodes of the same patient, facilitating longitudinal studies.

[ ]:
```
print(admissions.shape)
display(admissions.head())
```

```
(58976, 19)
```

	ROW_ID	SUBJECT_ID	HADM_ID	ADMITTIME	DISCHTIME	\
0	21	22	165315	2196-04-09 12:26:00	2196-04-10 15:54:00	

```
1      22          23   152223  2153-09-03 07:15:00  2153-09-08 19:10:00
2      23          23   124321  2157-10-18 19:34:00  2157-10-25 14:00:00
3      24          24   161859  2139-06-06 16:14:00  2139-06-09 12:48:00
4      25          25   129635  2160-11-02 02:06:00  2160-11-05 14:55:00

  DEATHTIME ADMISSION_TYPE         ADMISSION_LOCATION  \
0       NaN      EMERGENCY       EMERGENCY ROOM ADMIT
1       NaN       ELECTIVE  PHYS REFERRAL/NORMAL DELI
2       NaN      EMERGENCY  TRANSFER FROM HOSP/EXTRAM
3       NaN      EMERGENCY  TRANSFER FROM HOSP/EXTRAM
4       NaN      EMERGENCY       EMERGENCY ROOM ADMIT

          DISCHARGE_LOCATION INSURANCE LANGUAGE           RELIGION  \
0  DISC-TRAN CANCER/CHLDRN H   Private      NaN       UNOBTAINABLE
1           HOME HEALTH CARE  Medicare      NaN           CATHOLIC
2           HOME HEALTH CARE  Medicare     ENGL           CATHOLIC
3                       HOME   Private      NaN  PROTESTANT QUAKER
4                       HOME   Private      NaN       UNOBTAINABLE

  MARITAL_STATUS ETHNICITY            EDREGTIME            EDOUTTIME  \
0        MARRIED     WHITE  2196-04-09 10:06:00  2196-04-09 13:24:00
1        MARRIED     WHITE                  NaN                  NaN
2        MARRIED     WHITE                  NaN                  NaN
3         SINGLE     WHITE                  NaN                  NaN
4        MARRIED     WHITE  2160-11-02 01:01:00  2160-11-02 04:27:00

                                        DIAGNOSIS  HOSPITAL_EXPIRE_FLAG  \
0                          BENZODIAZEPINE OVERDOSE                     0
1  CORONARY ARTERY DISEASE\CORONARY ARTERY BYPASS…                     0
2                                      BRAIN MASS                     0
3                    INTERIOR MYOCARDIAL INFARCTION                     0
4                           ACUTE CORONARY SYNDROME                     0

   HAS_CHARTEVENTS_DATA
0                     1
1                     1
2                     1
3                     1
4                     1
```

0.4 ICU Stay Records and Critical Care Timeline

ICUSTAYS.csv describes all Intensive Care Unit (ICU) stays within a hospitalization event for each patient. It provides granular information on ICU admission and discharge times, type of care unit, and length of stay in ICU (LOS), which represents the primary target variable of this project. The dataset plays a central role in predicting patient outcomes and understanding care delivery within the ICU environment. The data can be linked with other clinical data sources to extract temporal trends and treatment patterns during the ICU stay.

[ ]:
```
print(icustays.shape)
display(icustays.head())
```

```
(61532, 12)
```

	ROW_ID	SUBJECT_ID	HADM_ID	ICUSTAY_ID	DBSOURCE	FIRST_CAREUNIT	LAST_CAREUNIT	FIRST_WARDID	LAST_WARDID	INTIME	OUTTIME	LOS
0	365	268	110404	280836	carevue	MICU	MICU	52	52	2198-02-14 23:27:38	2198-02-18 05:26:11	3.2490
1	366	269	106296	206613	carevue	MICU	MICU	52	52	2170-11-05 11:05:29	2170-11-08 17:46:57	3.2788
2	367	270	188028	220345	carevue	CCU	CCU	57	57	2128-06-24 15:05:20	2128-06-27 12:32:29	2.8939
3	368	271	173727	249196	carevue	MICU	SICU	52	23	2120-08-07 23:12:42	2120-08-10 00:39:04	2.0600
4	369	272	164716	210407	carevue	CCU	CCU	57	57	2186-12-25 21:08:04	2186-12-27 12:01:13	1.6202

0.5 Diagnosis Codes and Clinical Conditions Assigned During Admissions

DIAGNOSES_ICD.csv lists the diagnostic codes (ICD-9) assigned to each hospital admission, defining the clinical conditions for each patient encounter. D_ICD_DIAGNOSES.csv provides detailed descriptions of ICD-9 codes, offering the clinical context for each diagnosis. Together, these datasets are fundamental for identifying patient cohorts based on specific diseases, which is the starting point of the project pipeline. They enable comprehensive studies of comorbidity profiles and disease-specific clinical pathways of ICU patients.

[ ]:
```
print(diagnoses.shape)
display(diagnoses.head())
```

```
(651047, 5)
```

	ROW_ID	SUBJECT_ID	HADM_ID	SEQ_NUM	ICD9_CODE
0	1297	109	172335	1.0	40301
1	1298	109	172335	2.0	486
2	1299	109	172335	3.0	58281
3	1300	109	172335	4.0	5855
4	1301	109	172335	5.0	4254

[ ]:
```
print(d_icd_diagnoses.shape)
display(d_icd_diagnoses.head())
```

```
(14567, 4)
```

```
   ROW_ID ICD9_CODE              SHORT_TITLE  \
0     174     01166     TB pneumonia-oth test
1     175     01170    TB pneumothorax-unspec
2     176     01171   TB pneumothorax-no exam
3     177     01172  TB pneumothorx-exam unkn
4     178     01173  TB pneumothorax-micro dx

                                      LONG_TITLE
0  Tuberculous pneumonia [any form], tubercle bac…
1              Tuberculous pneumothorax, unspecified
2  Tuberculous pneumothorax, bacteriological or h…
3  Tuberculous pneumothorax, bacteriological or h…
4  Tuberculous pneumothorax, tubercle bacilli fou…
```

0.6 Continuous Monitoring of Physiological and Clinical Parameters in ICU

CHARTEVENTS.csv records bedside clinical observations and vital signs collected continuously during ICU stays. This is the largest dataset in MIMIC-III and contains high-resolution data on physiological measurements (e.g., heart rate, blood pressure, respiratory rate). It provides crucial insights into the acute clinical state of patients and allows for detailed time-series analysis to model patient deterioration or recovery trends. Due to its large size, it may be useful to sample it for exploratory data analysis.

[ ]:
```
print(chartevents.shape)
display(chartevents.head())
```

```
(100, 15)
```

```
   ROW_ID  SUBJECT_ID  HADM_ID  ICUSTAY_ID  ITEMID            CHARTTIME  \
0     788          36   165660      241249  223834  2134-05-12 12:00:00
1     789          36   165660      241249  223835  2134-05-12 12:00:00
2     790          36   165660      241249  224328  2134-05-12 12:00:00
3     791          36   165660      241249  224329  2134-05-12 12:00:00
4     792          36   165660      241249  224330  2134-05-12 12:00:00

             STORETIME   CGID   VALUE  VALUENUM VALUEUOM  WARNING  ERROR  \
0  2134-05-12 13:56:00  17525   15.00     15.00    L/min        0      0
1  2134-05-12 13:56:00  17525  100.00    100.00      NaN        0      0
2  2134-05-12 12:18:00  20823    0.37      0.37      NaN        0      0
3  2134-05-12 12:19:00  20823    6.00      6.00      min        0      0
4  2134-05-12 12:19:00  20823    2.50      2.50      NaN        0      0

   RESULTSTATUS  STOPPED
0           NaN      NaN
1           NaN      NaN
2           NaN      NaN
3           NaN      NaN
```

```
4           NaN      NaN
```

0.7 Laboratory Test Results and Biochemical Monitoring

LABEVENTS.csv documents the results of laboratory tests performed during hospital admissions. This dataset enables monitoring of biochemical and hematologic parameters over time, providing valuable information on organ function and systemic diseases. The data support the creation of predictive models based on the evolution of laboratory parameters in critically ill patients. As with CHARTEVENTS, sampling is recommended for initial exploration.

[ ]:
```
print(labevents.shape)
display(labevents.head())
```

```
(5000, 9)
```

	ROW_ID	SUBJECT_ID	HADM_ID	ITEMID	CHARTTIME	VALUE	VALUENUM	\
0	281	3	NaN	50820	2101-10-12 16:07:00	7.39	7.39	
1	282	3	NaN	50800	2101-10-12 18:17:00	ART	NaN	
2	283	3	NaN	50802	2101-10-12 18:17:00	-1	-1.00	
3	284	3	NaN	50804	2101-10-12 18:17:00	22	22.00	
4	285	3	NaN	50808	2101-10-12 18:17:00	0.93	0.93	

	VALUEUOM	FLAG
0	units	NaN
1	NaN	NaN
2	mEq/L	NaN
3	mEq/L	NaN
4	mmol/L	abnormal

0.8 Drug Administration and Fluid Management in ICU

INPUTEVENTS_MV.csv and INPUTEVENTS_CV.csv contain detailed information on fluids, drugs, and nutritional substances administered to patients during their ICU stay. They are essential for understanding treatment strategies and analyzing the effect of medication and fluid balance on patient outcomes. These datasets allow modeling of dose-response relationships and exploring the relationship between therapeutic interventions and ICU length of stay.

[ ]:
```
print(inputevents_mv.shape)
display(inputevents_mv.head())
```

```
(5000, 31)
```

	ROW_ID	SUBJECT_ID	HADM_ID	ICUSTAY_ID	STARTTIME	\
0	241	27063	139787	223259	2133-02-05 06:29:00	
1	242	27063	139787	223259	2133-02-05 05:34:00	
2	243	27063	139787	223259	2133-02-05 05:34:00	
3	244	27063	139787	223259	2133-02-03 12:00:00	
4	245	27063	139787	223259	2133-02-03 12:00:00	

	ENDTIME	ITEMID	AMOUNT	AMOUNTUOM	RATE	…	\

```
0  2133-02-05 08:45:00  225166    6.774532       mEq        NaN  …
1  2133-02-05 06:30:00  225944   28.132997        ml  30.142497  …
2  2133-02-05 06:30:00  225166    2.813300       mEq        NaN  …
3  2133-02-03 12:01:00  225893    1.000000      dose        NaN  …
4  2133-02-03 12:01:00  220949  100.000000        ml        NaN  …

  TOTALAMOUNTUOM ISOPENBAG  CONTINUEINNEXTDEPT  CANCELREASON  \
0             ml         0                   0             1
1             ml         0                   0             0
2             ml         0                   0             0
3             ml         0                   0             2
4             ml         0                   0             2

   STATUSDESCRIPTION COMMENTS_EDITEDBY COMMENTS_CANCELEDBY  \
0          Rewritten               NaN                  RN
1    FinishedRunning               NaN                 NaN
2    FinishedRunning               NaN                 NaN
3          Rewritten                RN                 NaN
4          Rewritten                RN                 NaN

         COMMENTS_DATE ORIGINALAMOUNT  ORIGINALRATE
0  2133-02-05 12:52:00      10.000000      0.050000
1                  NaN      28.132998     30.255817
2                  NaN       2.813300      0.050426
3  2133-02-03 17:06:00       1.000000      1.000000
4  2133-02-03 17:06:00     100.000000      0.000000

[5 rows x 31 columns]
```

[ ]:
```
print(inputevents_cv.shape)
display(inputevents_cv.head())
```

```
(5000, 22)

   ROW_ID  SUBJECT_ID   HADM_ID  ICUSTAY_ID            CHARTTIME  ITEMID  \
0     592       24457  184834.0    205776.0  2193-09-11 09:00:00   30056
1     593       24457  184834.0    205776.0  2193-09-11 12:00:00   30056
2     594       24457  184834.0    205776.0  2193-09-11 16:00:00   30056
3     595       24457  184834.0    205776.0  2193-09-11 19:00:00   30056
4     596       24457  184834.0    205776.0  2193-09-11 21:00:00   30056

   AMOUNT AMOUNTUOM  RATE  RATEUOM  …  ORDERID  LINKORDERID  STOPPED  \
0   100.0        ml   NaN      NaN  …   756654      9359133      NaN
1   200.0        ml   NaN      NaN  …  3564075      9359133      NaN
2   160.0        ml   NaN      NaN  …   422646      9359133      NaN
3   240.0        ml   NaN      NaN  …  5137889      9359133      NaN
4    50.0        ml   NaN      NaN  …  8343792      9359133      NaN
```

```
   NEWBOTTLE ORIGINALAMOUNT  ORIGINALAMOUNTUOM  ORIGINALROUTE ORIGINALRATE  \
0        NaN            NaN                 ml           Oral          NaN
1        NaN            NaN                 ml           Oral          NaN
2        NaN            NaN                 ml           Oral          NaN
3        NaN            NaN                 ml           Oral          NaN
4        NaN            NaN                 ml           Oral          NaN

  ORIGINALRATEUOM  ORIGINALSITE
0             NaN           NaN
1             NaN           NaN
2             NaN           NaN
3             NaN           NaN
4             NaN           NaN

[5 rows x 22 columns]
```

0.9 Fluid Output and Balance Data for ICU Patients

OUTPUTEVENTS.csv captures all recorded patient outputs (e.g., urine volume, drainage fluids) during ICU stays. This dataset is important for calculating fluid balance, which is a key clinical parameter associated with mortality and ICU length of stay. The information helps in understanding renal function and the physiological response to treatments administered during critical care.

[ ]:
```
print(outputevents.shape)
display(outputevents.head())
```

```
(5000, 13)
```

```
   ROW_ID  SUBJECT_ID   HADM_ID  ICUSTAY_ID            CHARTTIME  ITEMID  \
0     344       21219  177991.0    225765.0  2142-09-08 10:00:00   40055
1     345       21219  177991.0    225765.0  2142-09-08 12:00:00   40055
2     346       21219  177991.0    225765.0  2142-09-08 13:00:00   40055
3     347       21219  177991.0    225765.0  2142-09-08 14:00:00   40055
4     348       21219  177991.0    225765.0  2142-09-08 16:00:00   40055

   VALUE VALUEUOM            STORETIME   CGID  STOPPED  NEWBOTTLE  ISERROR
0  200.0       ml  2142-09-08 12:08:00  17269      NaN        NaN      NaN
1  200.0       ml  2142-09-08 12:08:00  17269      NaN        NaN      NaN
2  120.0       ml  2142-09-08 13:39:00  17269      NaN        NaN      NaN
3  100.0       ml  2142-09-08 16:17:00  17269      NaN        NaN      NaN
4  200.0       ml  2142-09-08 16:17:00  17269      NaN        NaN      NaN
```

0.10 Disease Cohort Selection

At this stage, it is possible to define the target population for my study. Because MIMIC-III is a very large dataset covering several diseases and types of patients, the project requires the selection of a specific subset of patients with a clinically well-defined disease. This approach reflects real-world clinical research, where models are usually developed and validated on homogeneous patient cohorts. Steps to achieve this goal are as follows.

0.10.1 Creating a Category Column

Create the ICD9_CATEGORY column by extracting the first 3 digits of the ICD9_CODE code. The ICD-9 codes used in MIMIC-III are very detailed, sometimes with hundreds of subcategories for the same clinical condition. By considering only the first 3 digits in a new column called ICD9_CATEGORY, related codes can be grouped together and a broader, more clinically meaningful categorization of diseases can be obtained. This greatly simplifies the cohort selection process.

[ ]:
```
diagnoses['ICD9_CATEGORY'] = diagnoses['ICD9_CODE'].astype(str).str[:3]
diagnoses.head() # Useful for grouping similar clinical conditions
```

[ ]:

	ROW_ID	SUBJECT_ID	HADM_ID	SEQ_NUM	ICD9_CODE	ICD9_CATEGORY
0	1297	109	172335	1.0	40301	403
1	1298	109	172335	2.0	486	486
2	1299	109	172335	3.0	58281	582
3	1300	109	172335	4.0	5855	585
4	1301	109	172335	5.0	4254	425

0.10.2 Retrieving Long Medical Description of Each ICD9 Code

Numeric ICD9 codes are difficult to interpret on their own. By merging DIAGNOSES_ICD with D_ICD_DIAGNOSES, one can better understand the medical meaning of each code and accurately select the target disease category for my study.

[ ]:
```
# Create ICD9_CATEGORY in D_ICD_DIAGNOSES to simplify the merge, resulting in a␣
 ↪dataset having a title for every code.
d_icd_diagnoses['ICD9_CATEGORY'] = d_icd_diagnoses['ICD9_CODE'].astype(str).
 ↪str[:3]
category_titles = d_icd_diagnoses[['ICD9_CATEGORY', 'LONG_TITLE']].
 ↪drop_duplicates()
diagnoses = diagnoses.merge(category_titles, on='ICD9_CATEGORY', how='left')

diagnoses.head()
```

[ ]:

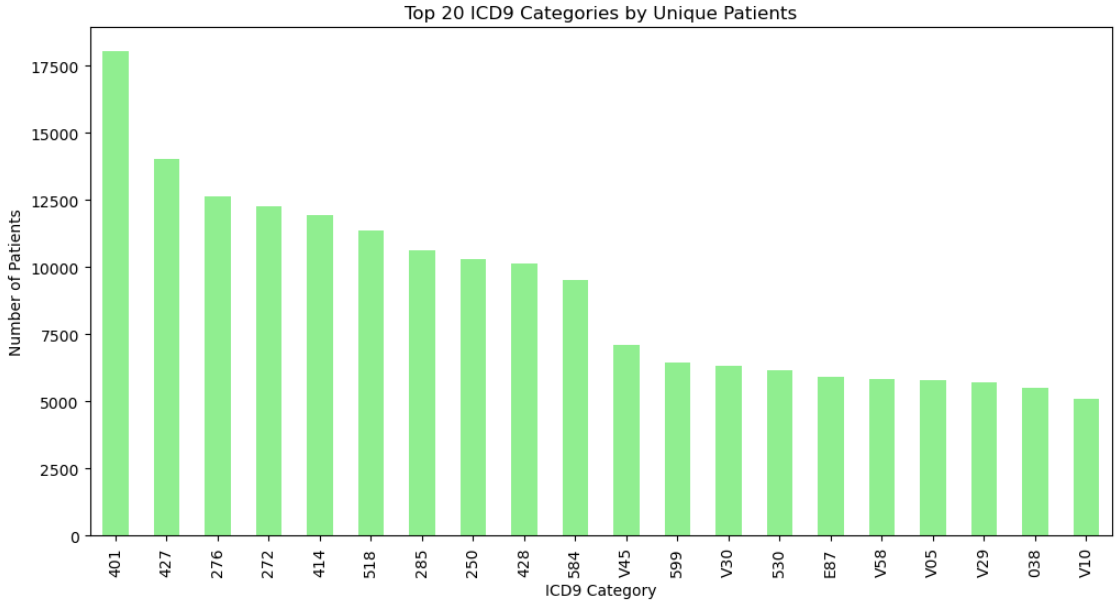
	ROW_ID	SUBJECT_ID	HADM_ID	SEQ_NUM	ICD9_CODE	ICD9_CATEGORY	\
0	1297	109	172335	1.0	40301	403	
1	1297	109	172335	1.0	40301	403	
2	1297	109	172335	1.0	40301	403	
3	1297	109	172335	1.0	40301	403	
4	1297	109	172335	1.0	40301	403	

	LONG_TITLE
0	Hypertensive chronic kidney disease, malignant…
1	Hypertensive chronic kidney disease, malignant…
2	Hypertensive chronic kidney disease, benign, w…
3	Hypertensive chronic kidney disease, benign, w…
4	Hypertensive chronic kidney disease, unspecifi…

0.10.3 Exploring and Selecting the Target Disease Cohort

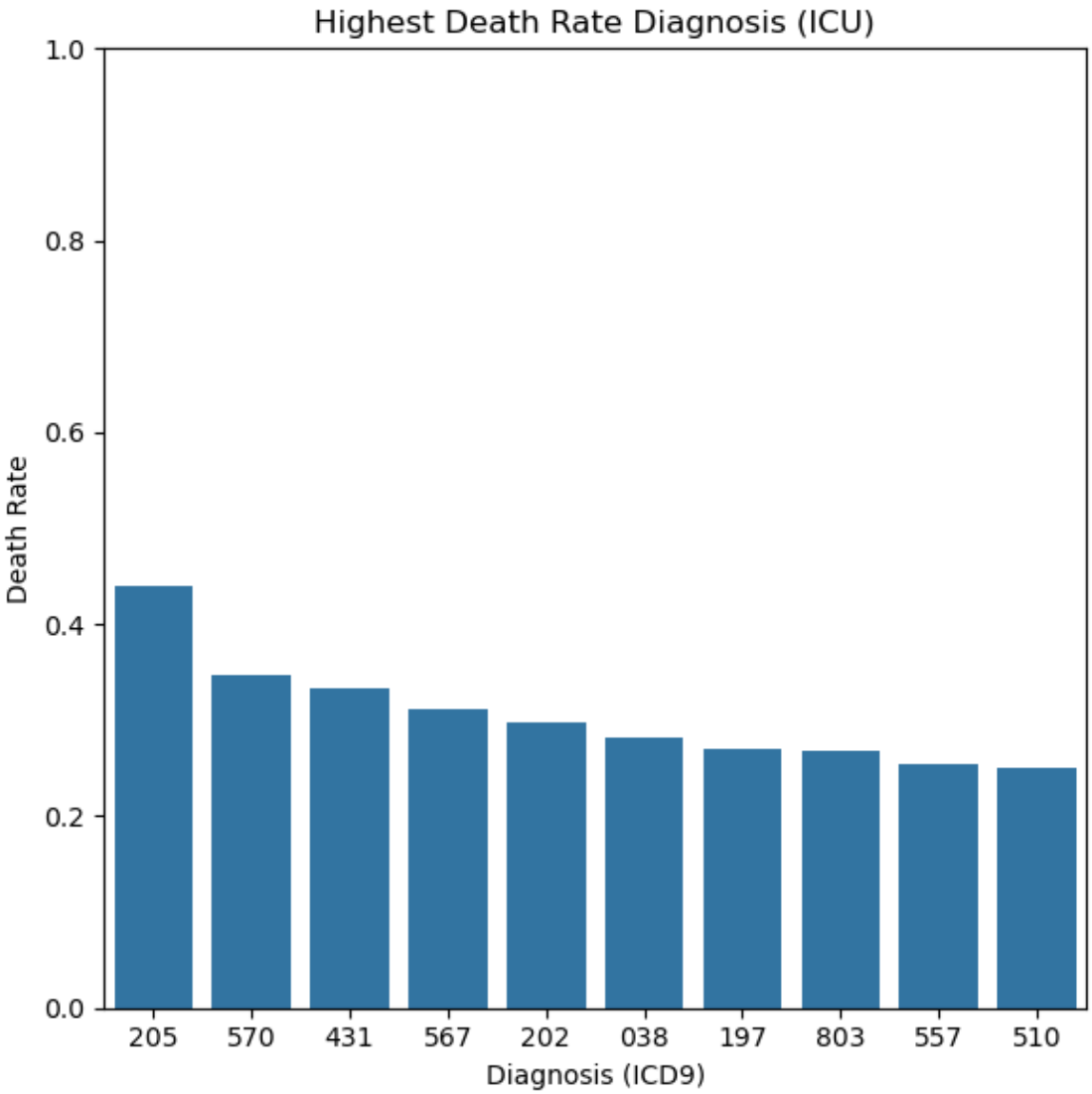
Patients Analysis In the plot showing the number of unique patients per ICD-9 category, code 038 (Sepsis) does not appear among the most prevalent overall—being surpassed, for example, by codes 250 (Diabetes Mellitus) and 414 (Coronary Artery Disease). Nonetheless, it stands out due to several clinically and epidemiologically relevant features. Specifically, code 038: it ranks among the top ten diagnoses in terms of ICU admissions and demonstrates a meaningful combination of clinical severity and cohort size By comparison: Code 250 is highly prevalent, but many associated admissions are non-critical or reflect secondary diagnoses; Code 414 is often managed outside of the ICU and more commonly appears as a comorbidity rather than the primary reason for admission. Conversely, code 038 represents a high-priority clinical condition in the ICU setting, being both sufficiently specific to avoid diagnostic ambiguity and statistically well-represented. These characteristics make it a strong candidate for developing predictive models of hospital length of stay.

[ ]:
```
diagnoses['ICD9_CATEGORY'] = diagnoses['ICD9_CODE'].astype(str).str[:3]
counts = diagnoses.groupby('ICD9_CATEGORY')['SUBJECT_ID'].nunique().
 ↪sort_values(ascending=False)

counts.head(20).plot(kind='bar', figsize=(12,6), color=['lightgreen'])
plt.title('Top 20 ICD9 Categories by Unique Patients')
plt.xlabel('ICD9 Category')
plt.ylabel('Number of Patients')
plt.savefig(ASSETS_PATH + 'top20_icd9_categories_by_unique_patients.png',␣
 ↪dpi=300)
plt.show()
```



Death Rate Analysis In the "Highest Death Rate Diagnosis (ICU)" plot, ICD-9 code 038 (Sepsis) exhibits a high mortality rate, surpassed only by codes 486 (Pneumonia) and 410 (Acute Myocardial Infarction). However, each of these alternatives presents specific limitations: Code 486 is marked by significant clinical heterogeneity and high variability in length of stay (LOS), which complicates the development of reliable predictive models. Code 410, despite its clinical severity, is generally associated with shorter and more variable LOS, often driven by rapid outcomes (either recovery or death), limiting the predictive utility of LOS-based models. In contrast, code 038 represents a high-risk condition with a more gradual clinical course and a sufficiently informative LOS distribution. This makes it particularly well-suited for predictive modeling focused on hospital length of stay estimation.

[ ]:
```
main_diag = diagnoses.sort_values('SEQ_NUM').drop_duplicates('HADM_ID',␣
↪keep='first')
main_diag['ICD9_CODE'] = main_diag['ICD9_CODE'].astype(str).str[:3]

diag_mortality = main_diag.merge(admissions, on='HADM_ID', how='left')

mortality_summary = diag_mortality.groupby('ICD9_CODE').agg(
    N_PATIENTS=('HADM_ID', 'count'),
    N_DEATHS=('HOSPITAL_EXPIRE_FLAG', 'sum')
).reset_index()

mortality_summary['DEATH_RATE'] = mortality_summary['N_DEATHS'] /␣
↪mortality_summary['N_PATIENTS']

mortality_summary = mortality_summary[mortality_summary['N_PATIENTS'] >= 50] #␣
↪Filter
top_mortalità = mortality_summary.sort_values('DEATH_RATE', ascending=False).
↪head(10) # # Sort by death rate

# Plot
plt.figure(figsize=(6, 6))
sns.barplot(data=top_mortalità, x='ICD9_CODE', y='DEATH_RATE')
plt.title('Highest Death Rate Diagnosis (ICU)')
plt.xlabel('Diagnosis (ICD9)')
plt.ylabel('Death Rate')
plt.ylim(0, 1)
plt.tight_layout()
plt.savefig(ASSETS_PATH + 'death_rate_by_diagnosis.png', dpi=300)
plt.show()
```



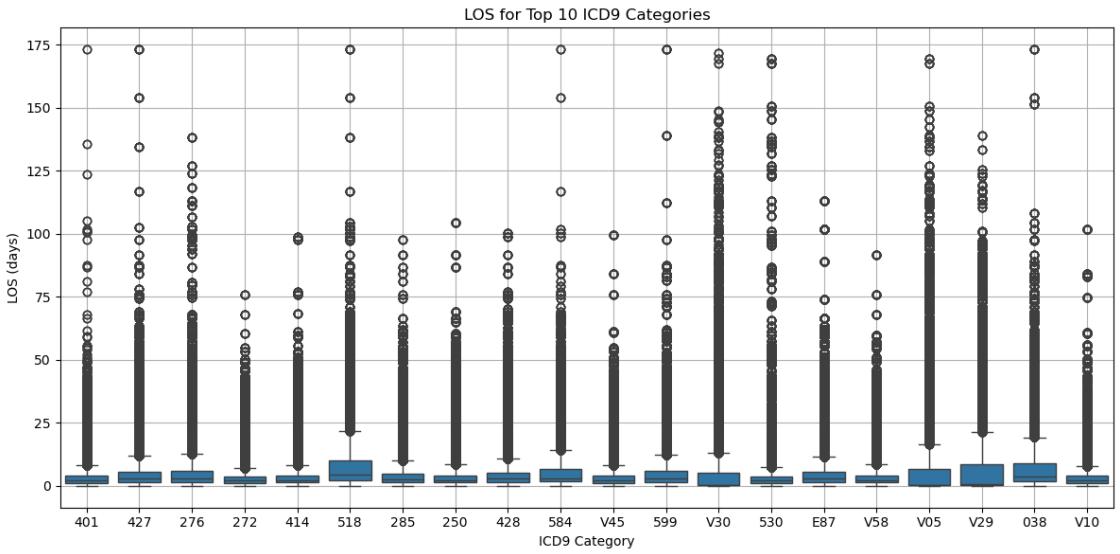
LOS Analysis The initial analysis focuses on intensive care unit (ICU) length of stay (LOS) across the most prevalent ICD-9 diagnostic categories and ICU admission diagnoses. Among these, category 038, corresponding to sepsis, stands out due to its moderate-to-high LOS distribution, with a substantial proportion of patients experiencing ICU stays longer than seven days. A particularly noteworthy feature is the relatively low dispersion in LOS for this group, especially when compared to other high-variability critical conditions such as acute respiratory failure (code 518) and acute hepatic failure (code 570). This reduced variability suggests that LOS in septic patients is more stable and predictable, which is a crucial property for the development of reliable predictive models, as it limits unexplained variance and enhances model robustness.

Overall, the findings indicate that septic patients: * Require considerable intensive care resources; * Represent a relatively homogeneous population with respect to LOS

This balance between a high median LOS and constrained variability makes sepsis an optimal

clinical condition for training and validating predictive models aimed at forecasting ICU stay and optimizing critical care resource allocation.

```
[ ]: counts = diagnoses.groupby('ICD9_CATEGORY')['SUBJECT_ID'].nunique().
      ↪sort_values(ascending=False).head(20).index

     # Filter
     diag_top = diagnoses[diagnoses['ICD9_CATEGORY'].isin(counts)]
     diag_top = diag_top.merge(icustays[['SUBJECT_ID', 'HADM_ID', 'ICUSTAY_ID',␣
      ↪'LOS']], on=['SUBJECT_ID', 'HADM_ID'], how='left')
     diag_top = diag_top.dropna(subset=['LOS'])

     # Plot
     plt.figure(figsize=(12, 6))
     sns.boxplot(x='ICD9_CATEGORY', y='LOS', data=diag_top, order=counts)
     plt.title('LOS for Top 10 ICD9 Categories')
     plt.xlabel('ICD9 Category')
     plt.ylabel('LOS (days)')
     plt.grid(True)
     plt.tight_layout()
     plt.savefig(ASSETS_PATH + 'top10_categories_los.png', dpi=300)
     plt.show()
```

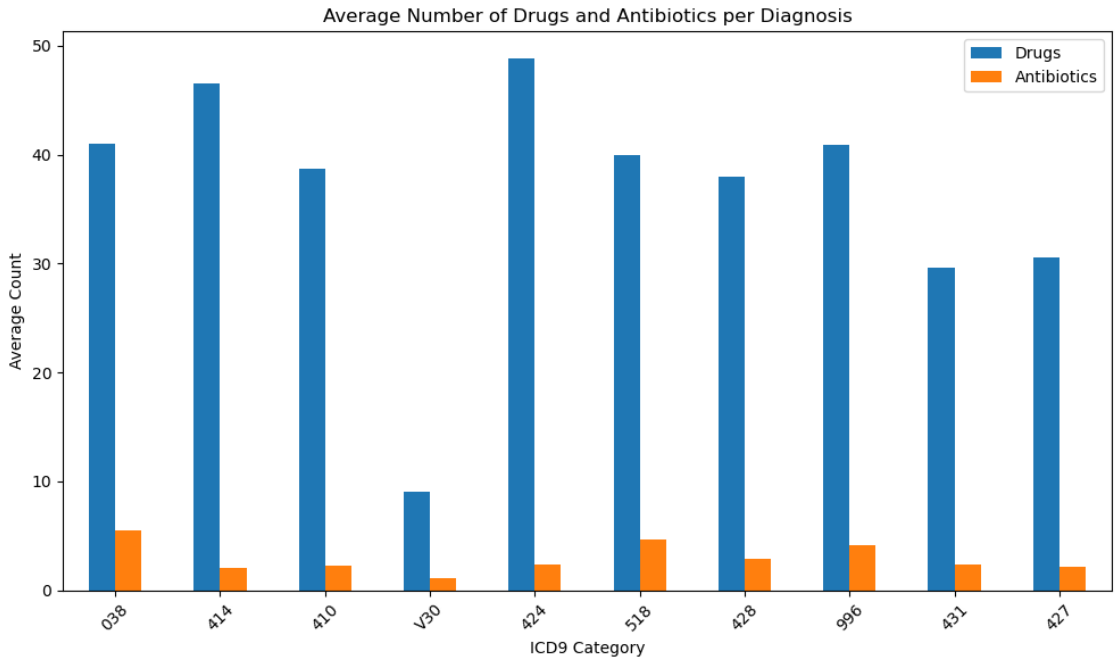


Treatments Analysis Analysis of the drug and antibiotic usage table highlights ICD-9 code 038 (Sepsis) as notable for having one of the highest average numbers of medications administered per patient (~38.6), and an high rate of antibiotic usage (~5.2), second only to code 486. These indicators reflect a high level of therapeutic intensity and provide a rich source of information - in terms of drug types and dosage patterns - for predictive modeling. The clinical complexity associated with sepsis appears sufficient to justify advanced analysis of length of stay (LOS). Conversely, codes 518

and 571 were excluded despite an even greater drug burden, due to their frequent association with multi-organ failure and highly variable interventions, which reduce the reliability and predictive accuracy of modeling efforts.

[ ]:
```
# Antibiotics labels
antibiotic_pattern = 'cillin|mycin|cef|penem|cycline|azole|floxacin'

# Counts drugs per HADM_ID
drug_counts = prescriptions.groupby('HADM_ID')['DRUG'].nunique().
 ↪reset_index(name='N_DRUGS')

# Counts antibiotics per HADM_ID
abx_mask = prescriptions['DRUG'].str.contains(antibiotic_pattern, case=False,␣
 ↪na=False)
abx_counts = prescriptions[abx_mask].groupby('HADM_ID')['DRUG'].nunique().
 ↪reset_index(name='N_ANTIBIOTICS')

# Merge
drug_data = pd.merge(drug_counts, abx_counts, on='HADM_ID', how='left').
 ↪fillna(0)

# Add ICD9 Category
main_diagnoses = diagnoses.sort_values('SEQ_NUM').drop_duplicates('HADM_ID',␣
 ↪keep='first')
main_diagnoses['ICD9_CATEGORY'] = main_diagnoses['ICD9_CODE'].astype(str).str[:
 ↪3]

# Add drugs counter
drug_data = pd.merge(drug_data, main_diagnoses[['HADM_ID', 'ICD9_CATEGORY']],␣
 ↪on='HADM_ID', how='left')

# Summary
summary = drug_data.groupby('ICD9_CATEGORY').agg({
    'N_DRUGS': 'mean',
    'N_ANTIBIOTICS': 'mean',
    'HADM_ID': 'count'
}).rename(columns={'HADM_ID': 'Patients', 'N_DRUGS': 'Drugs', 'N_ANTIBIOTICS':␣
 ↪'Antibiotics'}).reset_index()

summary = summary.sort_values(by='Patients', ascending=False).head(10)
summary.plot(x='ICD9_CATEGORY', y=['Drugs', 'Antibiotics'], kind='bar',␣
 ↪figsize=(10,6))
plt.title('Average Number of Drugs and Antibiotics per Diagnosis')
plt.ylabel('Average Count')
plt.xlabel('ICD9 Category')
plt.xticks(rotation=45)
plt.tight_layout()
```

```
plt.savefig(ASSETS_PATH + 'drug_antibiotic_usage.png', dpi=300)
plt.show()
```



Final Justification for Selecting ICD-9 Code 038 – Sepsis The multidimensional analysis of the main ICD-9 diagnoses identifies 038 – Septicemia/Sepsis as the most robust, balanced, and clinically relevant cohort for the task of predicting ICU length of stay (LOS). Unlike other diagnoses that excel in isolated dimensions but fall short in others, code 038 demonstrates a well-rounded profile across all key criteria:

Representativeness: It consistently ranks among the top ten ICD-9 codes in terms of patient volume and ICU admissions, ensuring statistical power without being overly generic or ambiguous, as is the case with code 250 (Diabetes Mellitus). Clinical Severity: It is associated with high—but not extreme—mortality, indicating a complex and sufficiently prolonged clinical course suitable for predictive modeling (unlike 410 – Acute Myocardial Infarction, which often involves rapid outcomes). Length of Stay (LOS): It exhibits a relatively long median LOS with low variability, an ideal scenario for building reliable predictive models. In contrast, codes such as 518 or 570 show extreme dispersion, which hinders modeling precision. Informational Richness: Patients diagnosed with sepsis typically receive numerous medications, including antibiotics, resulting in a structured and informative dataset highly conducive to training machine learning models. No other ICD-9 category analyzed meets all four criteria with equivalent robustness. Diagnoses like 486 (Pneumonia) or 570 (Liver Failure) offer specific advantages but are limited by structural weaknesses (e.g., clinical heterogeneity, low prevalence, or excessive variability), which undermine their predictive reliability. Therefore, selecting code 038 as the primary cohort allows for an optimal balance between statistical robustness, clinical relevance, and data richness, positioning it as the most suitable candidate for developing effective, generalizable, and clinically meaningful predictive models.

Criterion	038 – Sepsis	250 – Diabetes	410 – MI	518 – Resp. Failure
Patient Volume	High	Very High	High	Moderate
Clinical Severity	High	Low	Very High	Very High
LOS Stability	Moderate & Stable	Low & Variable	Short LOS	Very Variable
Treatment Informativeness	High (drugs & abx)	Low	Low	High but heterogeneous
Overall Suitability	Optimal	Weak context	Limited	Too noisy

0.10.4 Exporting the Sepsis Cohort for Downstream Use

After selecting the target population based on ICD-9 code 038 (sepsis), it is crucial to persistently save the list of admissions (HADM_ID) and patients (SUBJECT_ID) associated with this diagnosis. This step enables reusability in the next phases of the project (clinical visualization and feature engineering), ensuring reproducibility and consistency. The export also includes ICUSTAY_ID, when available, to simplify the join with temporal clinical events.

[ ]:
```
# Extract first diagnosis per admission
main_diagnoses = diagnoses.sort_values('SEQ_NUM').drop_duplicates('HADM_ID',␣
 ↪keep='first')

# Filter for Sepsis (ICD9_CATEGORY = '038')
sepsis = main_diagnoses[main_diagnoses['ICD9_CATEGORY'] == '038']

# Retain only identifiers
sepsis_ids = sepsis[['SUBJECT_ID', 'HADM_ID']].drop_duplicates()

# (Optional) Join with ICU stay IDs for completeness
icustays = pd.read_csv(PATH + "ICUSTAYS.csv", usecols=["SUBJECT_ID", "HADM_ID",␣
 ↪"ICUSTAY_ID"])
sepsis_ids = pd.merge(sepsis_ids, icustays, on=["SUBJECT_ID", "HADM_ID"],␣
 ↪how="left")

# Save to processed directory
sepsis_ids.to_csv("../data/processed/sepsis_cohort.csv", index=False)

# Confirm
print("[SUCCESS] Sepsis cohort exported:", sepsis_ids.shape)
sepsis_ids.head()
```

```
[SUCCESS] Sepsis cohort exported: (3686, 3)
```

[ ]:

	SUBJECT_ID	HADM_ID	ICUSTAY_ID
0	51797	104616	265369.0
1	44534	183659	204918.0

```
2        14828    144708    293475.0
3        14828    125239    288771.0
4        44500    101872    260996.0
```

The output indicates that 3686 hospital admissions (HADM_ID) were identified with a primary diagnosis of sepsis (ICD9 = 038), spanning multiple SUBJECT_ID values—some patients (e.g., 14828) appear more than once due to recurrent admissions. The ICUSTAY_ID column provides the corresponding ICU stay, confirming that the cohort has been successfully linked to intensive care episodes. This level of detail is essential for clinical time series analysis, as dynamic data in MIMIC-III (e.g., CHARTEVENTS, INPUTEVENTS_MV) are indexed by ICUSTAY_ID.

[ ]:
```
# 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/01_Data_Exploration.ipynb" ./ &> /
 ↪dev/null

# Then you can run the converter.

!jupyter nbconvert --to PDF "01_Data_Exploration.ipynb" &> /dev/null
```

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
```

03_EDA

June 8, 2025

1 EDA

1.1 EDA Setup: Visualization Style and Histogram Function

The first block of the Exploratory Data Analysis chapter defines essential configurations for reproducible and aesthetically consistent plotting. It establishes standardized paths for accessing processed data (`EXPORT_PATH`) and for saving visualization assets (`ASSETS_PATH`), following the principle of separation between raw computation and derived outputs.

The plotting environment is configured with `seaborn`'s `"whitegrid"` style, which facilitates readability in scientific plots. `matplotlib`'s global figure size is adjusted to ensure uniform layout across different visualizations.

Custom Histogram Plot Function: The function `plot_histogram()` is designed to produce high-quality histograms enriched with kernel density estimation (`kde`) by default. It allows for flexible customization of:

- **Binning** (`bins`)
- **Labels and titles**
- **Figure size**
- **Automatic file saving** (controlled by the `save_path` argument)

This modular utility function enhances code reusability and encourages consistent formatting throughout the EDA chapter, which is particularly valuable in a thesis-level project that emphasizes clarity and visual insight.

[ ]:
```
EXPORT_PATH = "../data/processed/"
ASSETS_PATH = "../assets/plots/eda/"

import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
import os

# === Plot Style ===
sns.set(style="whitegrid")
plt.rcParams["figure.figsize"] = (10, 6)
def plot_histogram(
    data, column, bins=30, kde=True, figsize=(10, 4),
```

```
    title=None, xlabel=None, ylabel="Number of Patients",
    save_path=None
):
    plt.figure(figsize=figsize)
    sns.histplot(data[column], bins=bins, kde=kde)
    plt.title(title if title else f"{column} Distribution")
    plt.xlabel(xlabel if xlabel else column)
    plt.ylabel(ylabel)
    plt.tight_layout()
    if save_path:
        plt.savefig(save_path)
    plt.show()
```

1.1.1 Dataset Loading and Structural Sanity Check

Before any statistical or visual exploration can be performed, the dataset `df_final_static.csv` is reloaded from disk to ensure isolation between the data preparation and EDA stages. This practice enhances modularity, reproducibility, and minimizes memory footprint across different execution environments (e.g., Jupyter kernels, pipelines).

The following diagnostics are executed to validate the structure and integrity of the dataset:

- `df_final.shape`: Displays the overall dimensions of the dataset, serving as a sanity check that no rows were inadvertently filtered or added since export.
- `df_final.head()`: A visual preview of the first few rows, useful for verifying column types, expected values, and possible categorical encodings.
- `df_final.columns`: Lists all column names, offering a quick overview of the available features for downstream analysis.
- `df_final.isnull().sum().sort_values(ascending=False)/len(df_final)`: Computes the proportion of missing values for each column. This step is essential for evaluating data quality and for guiding imputation strategies or exclusion decisions.

These steps collectively ensure that the dataset is in a clean and analyzable state, aligning with rigorous scientific standards for empirical research.

[ ]:
```
# === Load dataset ===
df_final = pd.read_csv(os.path.join(EXPORT_PATH, "df_final_static.csv"))

# === Confirm structure ===
print(df_final.shape)
df_final.head()
df_final.columns
df_final.isnull().sum().sort_values(ascending=False)/len(df_final)
```

```
(3685, 16)
```

[ ]:
```
SUBJECT_ID              0.0
HADM_ID                 0.0
ICUSTAY_ID              0.0
```

```
AGE                     0.0
GENDER                  0.0
ADMISSION_TYPE          0.0
ADMISSION_LOCATION      0.0
INSURANCE               0.0
FIRST_CAREUNIT          0.0
LOS                     0.0
HOSPITAL_EXPIRE_FLAG    0.0
INTIME_HOUR             0.0
INTIME_WEEKDAY          0.0
ADMITTIME_HOUR          0.0
ADMITTIME_WEEKDAY       0.0
INTIME                  0.0
dtype: float64
```

1.1.2 Descriptive Summary of Numeric Variables

This step provides a statistical snapshot of the numeric features within the dataset through the `describe().T` method, which transposes the default output to a column-wise orientation for enhanced readability.

For each numeric variable, the following summary statistics are computed:

- **Count**: Number of non-missing entries
- **Mean** and **Standard Deviation**: Indicators of central tendency and dispersion
- **Min, 25th, 50th (Median), 75th, and Max**: Useful for detecting skewness, spread, and potential outliers

This profiling phase is particularly valuable in clinical datasets like MIMIC-III, where variables such as age or ICU Length of Stay (LOS) often display right-skewed distributions, long tails, or discretized value spikes due to hospital policies (e.g., fixed discharge times).

By scanning these metrics, one can anticipate the need for transformations (e.g., log-scaling for LOS), outlier mitigation, and scaling adjustments in downstream modeling.

[ ]:
```
print("\n[INFO] Summary statistics for numeric variables:")
display(df_final.describe().T)
```

```
[INFO] Summary statistics for numeric variables:
```

	count	mean	std	min	\
SUBJECT_ID	3685.0	38042.643691	29519.241245	3.0000	
HADM_ID	3685.0	149043.439077	29176.674824	100074.0000	
ICUSTAY_ID	3685.0	250221.804885	28861.797019	200003.0000	
AGE	3685.0	68.258887	15.991439	0.0000	
LOS	3685.0	5.744356	7.677370	0.0079	
HOSPITAL_EXPIRE_FLAG	3685.0	0.287110	0.452475	0.0000	
INTIME_HOUR	3685.0	13.721574	7.062893	0.0000	
INTIME_WEEKDAY	3685.0	3.023338	1.988703	0.0000	

ADMITTIME_HOUR	3685.0	13.995387	7.084968	0.0000
ADMITTIME_WEEKDAY	3685.0	3.016554	2.011907	0.0000

	25%	50%	75%	max
SUBJECT_ID	13934.0000	27748.0000	62871.000	99985.0000
HADM_ID	123675.0000	148651.0000	175213.000	199943.0000
ICUSTAY_ID	225602.0000	250364.0000	275615.000	299950.0000
AGE	58.0000	70.0000	81.000	91.0000
LOS	1.7219	3.0194	6.602	97.2972
HOSPITAL_EXPIRE_FLAG	0.0000	0.0000	1.000	1.0000
INTIME_HOUR	8.0000	16.0000	20.000	23.0000
INTIME_WEEKDAY	1.0000	3.0000	5.000	6.0000
ADMITTIME_HOUR	9.0000	16.0000	20.000	23.0000
ADMITTIME_WEEKDAY	1.0000	3.0000	5.000	6.0000

1.2 Analysis of feature distributions

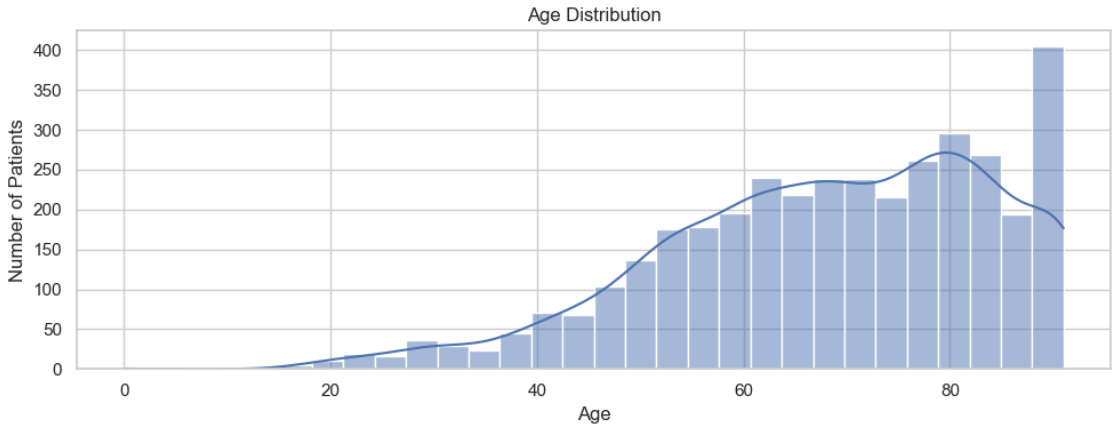
1.2.1 Age Distribution in ICU Sepsis Cohort

The histogram above illustrates the age distribution of patients admitted to the ICU with a diagnosis of sepsis. The distribution is right-skewed, with the majority of patients concentrated between the ages of 60 and 90. A significant spike is observed at age 91, which corresponds to the upper censoring limit imposed by the MIMIC-III dataset to preserve patient anonymity for individuals aged 89 and above.

This pattern is consistent with clinical expectations: elderly patients are more vulnerable to severe sepsis and are more frequently admitted to intensive care. The presence of a density tail in the lower age brackets (under 40) indicates that younger patients are present but far less frequent, likely representing cases of acute or atypical infections.

The distribution supports the decision to include **age as a primary predictor** in modeling ICU Length of Stay (LOS), as both biological resilience and comorbidity burden are age-dependent. Additionally, the sharp censoring at 91 must be taken into account to avoid bias or misinterpretation in models that assume a continuous age range.

[ ]:
```
plot_histogram(
    data=df_final,
    column="AGE",
    bins=30,
    title="Age Distribution",
    xlabel="Age",
    save_path=ASSETS_PATH + "age_distribution.png"
)
```



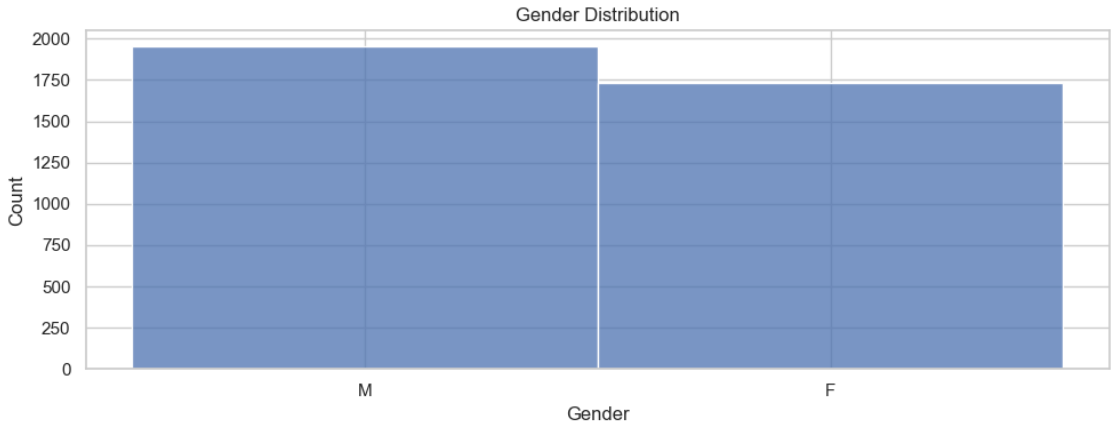
1.2.2 Gender Distribution in the ICU Sepsis Cohort

The bar chart illustrates the distribution of biological sex among ICU patients diagnosed with sepsis. The population consists of a slightly higher number of males (`M`) compared to females (`F`), with approximately 1950 male patients versus 1750 females.

This modest male predominance aligns with clinical literature suggesting that males are more frequently affected by sepsis, potentially due to differences in immune response, comorbidities, and healthcare access. However, the distribution remains reasonably balanced, implying that gender-specific bias is unlikely to be a major concern in the downstream modeling process.

It is important to note that while gender may not have a strong predictive signal on its own, it can interact with other variables (e.g., age, admission type, comorbidities) in non-linear ways. As such, it remains a useful covariate to retain in the model, especially when exploring explainability or fairness.

[ ]:
```
plot_histogram(
    data=df_final,
    column="GENDER",
    bins=len(df_final["GENDER"].unique()),
    kde=False,
    title="Gender Distribution",
    xlabel="Gender",
    ylabel="Count",
    save_path=ASSETS_PATH + "gender_distribution.png"
)
```



1.2.3 Distribution of ICU Length of Stay (LOS)

The histogram visualizes the empirical distribution of ICU Length of Stay (LOS), measured in days, for patients diagnosed with sepsis. As anticipated in clinical datasets, the distribution is **heavily right-skewed**, with the majority of stays concentrated in the 0–10 day range.

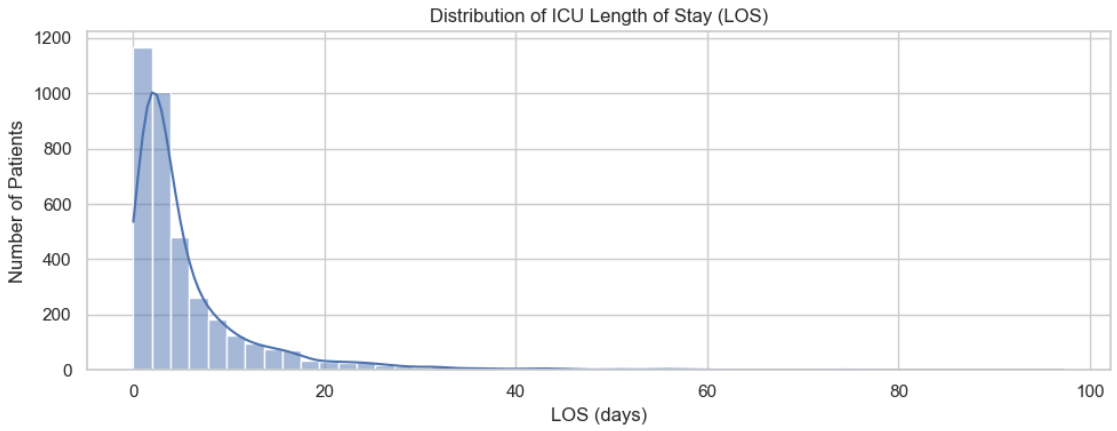
The tail extends considerably, with some extreme cases reaching up to ~100 days. Quantile statistics confirm this long-tail behavior:

- The **90th percentile** is at approximately **13.3 days**, indicating that 90% of patients are discharged within two weeks.
- The **99th percentile** is at **31.8 days**, suggesting that extreme long stays are rare but present.
- A total of **41 outliers** have a LOS exceeding **60 days**, representing clinically exceptional cases that may reflect complications, comorbidities, or institutional delays.

This pronounced asymmetry suggests that **log transformation** of LOS may be beneficial to stabilize variance and improve model performance. Furthermore, the presence of extreme outliers necessitates careful validation and may call for **robust modeling techniques** or outlier handling strategies during training.

[ ]:
```
plot_histogram(
    data=df_final,
    column="LOS",
    bins=50,
    title="Distribution of ICU Length of Stay (LOS)",
    xlabel="LOS (days)",
    save_path=ASSETS_PATH + "los_distribution.png"
)

q90 = df_final['LOS'].quantile(0.90)
q99 = df_final['LOS'].quantile(0.99)
max_los = df_final['LOS'].max()

print(f'Outliers: {df_final[(df_final.LOS>60)].shape[0]}')
```

```
print(f"90° percentile: {q90:.2f} days")
print(f"99° percentile: {q99:.2f} days")
print(f"Max LOS: {max_los:.2f} days")
```




```
Outliers: 7
90° percentile: 13.71 days
99° percentile: 37.41 days
Max LOS: 97.30 days
```

[ ]:
```
# Remove outliers
df_final = df_final[df_final['LOS'] <= 60]
```

[ ]:
```
# 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/03_EDA.ipynb" ./ &> /dev/null

# Then you can run the converter.

!jupyter nbconvert --to PDF "03_EDA.ipynb" &> /dev/null
```

04_Feature_Engineering

June 8, 2025

1 Feature Engineering

1.0.1 Setup for Feature Engineering

At the beginning of the feature engineering phase, the same plotting configuration and utility function used in EDA are retained. This ensures consistency in the visual inspection of transformed or newly constructed features, which is particularly valuable when validating assumptions or diagnosing feature quality.

The `plot_histogram()` function remains a central tool for assessing the distribution, modality, and potential skewness of both raw and derived features. As new variables are engineered—especially from dynamic tables (e.g., `CHARTEVENTS`, `INPUTEVENTS_MV`) or through aggregations (e.g., mean, std, skew)—visual confirmation becomes essential.

Maintaining visual standards across exploratory and engineering phases reflects good scientific rigor, supports reproducibility, and facilitates documentation for thesis-level reporting.

[ ]:
```
EXPORT_PATH = "../data/processed/"
ASSETS_PATH = "../assets/plots/eda/"

import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
import os

# === Plot Style ===
sns.set(style="whitegrid")
plt.rcParams["figure.figsize"] = (10, 6)
def plot_histogram(
    data, column, bins=30, kde=True, figsize=(10, 4),
    title=None, xlabel=None, ylabel="Number of Patients",
    save_path=None
):
    plt.figure(figsize=figsize)
    sns.histplot(data[column], bins=bins, kde=kde)
    plt.title(title if title else f"{column} Distribution")
    plt.xlabel(xlabel if xlabel else column)
    plt.ylabel(ylabel)
```

```
    plt.tight_layout()
    if save_path:
        plt.savefig(save_path)
    plt.show()
```

[ ]:
```
# === Load dataset ===
df_final = pd.read_csv(os.path.join(EXPORT_PATH, "df_final_static.csv"))

# === Confirm structure ===
print(df_final.shape)
df_final.head()
```

```
(3685, 16)
```

[ ]:

	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

1.1 Adding Dynamic Features (Temporal Aggregation)

This block initiates the temporal feature engineering process by mapping clinically relevant **vital signs** to their corresponding ITEMIDs in the MIMIC-III database, as per official documentation and clinical guidelines.

Each vital sign (e.g., Heart Rate, Systolic Blood Pressure, Temperature) may be recorded under multiple ITEMIDs due to differences in equipment, measurement protocols, or care units. Grouping these codes ensures that all valid measurements are captured uniformly across patients and time points.

The dictionary `vital_items` serves as a reference map, organizing ITEMIDs under semantic labels. The flattened `itemid_to_label` dictionary enables rapid reverse lookup from an individual ITEMID to its physiological label—a crucial step for categorizing and aggregating measurements in downstream steps.

This systematic mapping allows the CHARTEVENTS table, which is rich but messy, to be filtered and interpreted in a clinically coherent manner, transforming it from a semi-structured log to a set of analyzable features.

[ ]:
```
# Define ITEMIDs per MIMIC-III documentation for vital signs
vital_items = {
    "Heart Rate": [211, 220045],
    "Systolic BP": [51, 455, 220179, 220050],
    "Diastolic BP": [8368, 8441, 220180, 220051],
    "Mean BP": [52, 456, 220052],
    "Respiratory Rate": [618, 220210],
    "Temperature": [678, 223761],
    "SpO2": [646, 220277],
    "Glucose": [807, 220621]
}

itemid_to_label = {item: label for label, items in vital_items.items() for item␣
 ↪in items}
```

1.1.1 Temporal Filtering and Labeling of Vital Signs from CHARTEVENTS

This block transforms the raw `CHARTEVENTS` table—one of the most voluminous and granular tables in MIMIC-III—into a temporally-filtered and semantically-labeled set of measurements for feature engineering.

1. **Data Import and Merging**: `chartevents_sepsis.csv`, a filtered export of `CHARTEVENTS`, is joined with the ICU cohort on `ICUSTAY_ID`. This operation ensures that only ICU stays of interest (i.e., sepsis patients) are considered, and that the timestamp `INTIME` of each ICU admission is accessible for temporal alignment.
2. **Time Window Filtering (0–24h)**: A new variable `HOURS_FROM_INTIME` is computed to measure the number of hours elapsed from ICU admission to each recorded event. Only events occurring in the first 24 hours are retained. This window is clinically motivated: early vital sign patterns often serve as early warning signals and are crucial for predictive modeling.
3. **Item and Value Filtering**: Only events with a recognized `ITEMID` (from the `itemid_to_label` dictionary) and non-null `VALUENUM` are retained. This ensures semantic clarity and numerical integrity. Each event is then annotated with a `VITAL_TYPE`, enabling grouping and statistical aggregation in subsequent steps.

This pipeline transforms millions of event-level entries into a manageable and interpretable structure. It is both **clinically sound** and **computationally efficient**, paving the way for robust temporal feature engineering.

[ ]:
```
# Load CHARTEVENTS and cohort ICU
chartevents = pd.read_csv(EXPORT_PATH + "chartevents_sepsis.csv",␣
 ↪parse_dates=["CHARTTIME"])
cohort_icu = pd.read_csv(EXPORT_PATH + "df_final_static.csv")

# Join CHARTEVENTS with cohort ICU
first24h = chartevents.merge(cohort_icu[["ICUSTAY_ID", "INTIME"]],␣
 ↪on="ICUSTAY_ID", how="inner")
first24h["HOURS_FROM_INTIME"] = (first24h["CHARTTIME"] - pd.
 ↪to_datetime(first24h["INTIME"])).dt.total_seconds() / 3600
first24h = first24h[(first24h["HOURS_FROM_INTIME"] >= 0) &␣
 ↪(first24h["HOURS_FROM_INTIME"] <= 24)]

# Filter valid ITEMIDs and non-null VALUENUM
first24h = first24h[first24h["ITEMID"].isin(itemid_to_label)]
first24h = first24h[first24h["VALUENUM"].notnull()]
first24h["VITAL_TYPE"] = first24h["ITEMID"].map(itemid_to_label)
```

```
/var/folders/0j/nhv3j29j5bngf6nym9kpvhl80000gn/T/ipykernel_58489/3589101290.py:2
: DtypeWarning: Columns (5) have mixed types. Specify dtype option on import or
set low_memory=False.
  chartevents = pd.read_csv(EXPORT_PATH + "chartevents_sepsis.csv",
parse_dates=["CHARTTIME"])
```

1.1.2 Temporal Aggregation of Vital Signs in First 24 Hours

This block performs statistical aggregation of vital signs collected in the first 24 hours of ICU stay. These aggregations yield **hand-crafted features** that capture the distributional behavior of each physiological parameter over the early hours of critical illness.

1. **Grouping by VITAL_TYPE**: For each predefined vital sign (e.g., "Heart Rate", "SpO2"), the subset of data entries is filtered from `first24h` using the label from `VITAL_TYPE`.
2. **Statistical Aggregation**: For each ICU stay (`ICUSTAY_ID`), the following descriptive statistics are computed on the `VALUENUM` of the vital sign:
   - **mean**: central tendency
   - **std**: variation/spread
   - **min**/**max**: range
   - **count**: data availability (proxy for measurement density)
   - **skew**: asymmetry in the distribution
3. **Feature Naming**: Feature names are standardized using uppercase transformation and concatenation of the vital sign with the statistic (e.g., `HEART_RATE_MEAN`, `SPO2_STD`).
4. **Final Merge**: The resulting per-vital DataFrames are merged using outer joins on

ICUSTAY_ID, ensuring that missing values are preserved for downstream imputation rather than excluded prematurely.

The final df_vitals table contains one row per ICU stay, with one column per statistical property of each vital sign. This structured matrix is ready to be merged with the static cohort (df_final_static.csv) and used in modeling pipelines.

[ ]:
```
# Aggregation per ICUSTAY_ID and VITAL_TYPE
vital_features = []

for label in vital_items.keys():
    temp = first24h[first24h["VITAL_TYPE"] == label]
    stats = temp.groupby("ICUSTAY_ID")["VALUENUM"].agg(["mean", "std", "min",␣
 ↪"max", "count", "skew"]).reset_index()
    stats.columns = ["ICUSTAY_ID"] + [f"{label.upper().replace(' ', '_')}_{stat.
 ↪upper()}" for stat in stats.columns[1:]]
    vital_features.append(stats)

# Merge all together
from functools import reduce
df_vitals = reduce(lambda left, right: pd.merge(left, right, on="ICUSTAY_ID",␣
 ↪how="outer"), vital_features)
```

1.1.3 Final Dataset Assembly: Merging Static and Dynamic Features

In this final stage of feature engineering, the previously constructed temporal features (df_vitals)—derived from physiological signals measured during the first 24 hours of ICU admission—are merged with the static cohort dataset (df_final_static.csv), which contains demographic, admission, and administrative information. * **Merge Operation**: The join is performed on the ICUSTAY_ID key using a **left join**, which ensures that all records from the static dataset are preserved—even if some ICU stays have missing or incomplete time-series data. This design is essential for maintaining the full patient cohort and handling missing data explicitly during preprocessing.

- **Export for Downstream Tasks**: The final dataset df_final_enriched is saved to disk. It now includes both static attributes (e.g., age, gender, admission type) and temporal descriptors (e.g., mean and variability of heart rate, blood pressure, etc.), forming a **rich, multimodal feature space** ideal for supervised learning.

This unified dataset becomes the **central input** for the next phase—model training and validation—and represents a carefully engineered structure that reflects both clinical relevance and data integrity.

[ ]:
```
# Merge dynamic features into df_final
df_final = pd.read_csv(EXPORT_PATH + "df_final_static.csv")
df_enriched = df_final.merge(df_vitals, on="ICUSTAY_ID", how="left")

# Save to disk
df_enriched.to_csv(EXPORT_PATH + "df_final_enriched.csv", index=False)
```

```
print(f"Final enriched dataset shape: {df_enriched.shape}")
```

```
Final enriched dataset shape: (3685, 64)
```

1.2 Data Cleaning

1.2.1 Missing Data Profiling: Assessing Feature Completeness

This step performs a systematic assessment of missing data across the enriched dataset (`df_enriched`), with the goal of identifying features that may require imputation, exclusion, or special handling prior to model training.

By computing the proportion of `NaN` values for each feature and sorting the results in descending order, the analysis highlights which variables have the most severe completeness issues. Features with more than **10% missing values** are particularly critical, as they may:

- Bias model learning if left unaddressed
- Affect generalization if their distribution differs between train and test sets
- Reduce interpretability, especially in clinical contexts where data sparsity reflects operational constraints

Reporting the **total number of features** (`len(missing_data)`) provides an overview of the feature space size and supports the justification of future dimensionality reduction or feature selection strategies.

This diagnostic serves as the foundation for a rational and reproducible missing data handling policy—an essential component of any robust machine learning pipeline, particularly in healthcare applications.

[ ]:
```
# Print missing data statistics greater than 10%
missing_data = df_enriched.isnull().sum().sort_values(ascending=False) /␣
  ↪len(df_enriched)
print(missing_data)
print(len(missing_data))
```

```
GLUCOSE_SKEW          0.808412
MEAN_BP_SKEW          0.797829
MEAN_BP_STD           0.794301
MEAN_BP_COUNT         0.790231
MEAN_BP_MIN           0.790231
                          …
FIRST_CAREUNIT        0.000000
GENDER                0.000000
INSURANCE             0.000000
ADMISSION_LOCATION    0.000000
SUBJECT_ID            0.000000
Length: 64, dtype: float64
64
```

1.2.2 Feature Pruning Based on Missingness Threshold

In this step, variables with excessive proportions of missing values are systematically removed from the dataset. Specifically, features with more than **49% missing data** are discarded entirely, following the rationale that highly sparse variables contribute little to predictive power and may introduce instability during imputation or modeling.

Rationale for the 49% Threshold:

- Variables with over half their values missing lack sufficient representation to allow reliable learning of patterns.
- Retaining such features often results in **uninformative noise**, increased dimensionality, and increased variance in downstream models.
- By removing only the worst-offending variables, the procedure preserves the majority of potentially informative features while improving the dataset's statistical robustness.

The list of removed features is stored in `features_to_remove`, allowing traceability and reproducibility of preprocessing steps—an essential requirement in scientific data workflows.

The final dataset `df` now has improved completeness and is better suited for subsequent imputation and model training.

[ ]:
```
df = df_enriched.copy()
# Remove features with more than 60% missing data
features_to_remove = missing_data[missing_data > 0.49].index.tolist()
# Drop features with more than 60% missing data
df.drop(columns=features_to_remove, inplace=True)
# Print the number of features removed
len(features_to_remove)
```

[ ]: 26

1.2.3 Advanced Missing Value Imputation via Iterative Imputer (MICE)

To handle missing data in numerical features, this step employs the **Iterative Imputer** from `scikit-learn`, a sophisticated approach that models each incomplete feature as a function of the others in a **Bayesian regression-like framework**. This method—commonly referred to as MICE—is particularly advantageous in healthcare datasets where variable interdependence is high and simple imputation (e.g., mean or median) may fail to capture complex relationships.

- The dataset is first filtered to retain only numerical columns, ensuring that the imputer operates on continuous and discrete numeric values.
- The `IterativeImputer` is applied, estimating missing values by iteratively modeling each column using all other columns as predictors. This preserves the **multivariate distributional structure** of the dataset.
- The imputed matrix is then converted back into a `pandas` DataFrame with restored column names.
- If non-numeric columns (e.g., categorical variables) were excluded from imputation, they are reattached to the imputed frame using `pd.concat()`.

This method significantly enhances the robustness of the feature space by **preserving inter-feature correlations** and minimizing information loss, which is particularly important for downstream models sensitive to missingness, such as neural networks or tree ensembles.

[ ]:
```
from sklearn.experimental import enable_iterative_imputer  # noqa
from sklearn.impute import IterativeImputer
import pandas as pd

# Filtra solo colonne numeriche
df_numeric = df.select_dtypes(include='number')

# Imputazione iterativa
iter_imputer = IterativeImputer()
df_imputed = pd.DataFrame(
    iter_imputer.fit_transform(df_numeric),
    columns=df_numeric.columns
)

# Se vuoi reinserire le colonne non numeriche:
df_final = pd.concat([df_imputed, df.drop(columns=df_numeric.columns)], axis=1)

df_final.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 3685 entries, 0 to 3684
Data columns (total 38 columns):
 #   Column                  Non-Null Count  Dtype
---  ------                  --------------  -----
 0   SUBJECT_ID              3685 non-null   float64
 1   HADM_ID                 3685 non-null   float64
 2   ICUSTAY_ID              3685 non-null   float64
 3   AGE                     3685 non-null   float64
 4   LOS                     3685 non-null   float64
 5   HOSPITAL_EXPIRE_FLAG    3685 non-null   float64
 6   INTIME_HOUR             3685 non-null   float64
 7   INTIME_WEEKDAY          3685 non-null   float64
 8   ADMITTIME_HOUR          3685 non-null   float64
 9   ADMITTIME_WEEKDAY       3685 non-null   float64
 10  HEART_RATE_MEAN         3685 non-null   float64
 11  HEART_RATE_STD          3685 non-null   float64
 12  HEART_RATE_MIN          3685 non-null   float64
 13  HEART_RATE_MAX          3685 non-null   float64
 14  HEART_RATE_COUNT        3685 non-null   float64
 15  HEART_RATE_SKEW         3685 non-null   float64
 16  RESPIRATORY_RATE_MEAN   3685 non-null   float64
 17  RESPIRATORY_RATE_STD    3685 non-null   float64
 18  RESPIRATORY_RATE_MIN    3685 non-null   float64
 19  RESPIRATORY_RATE_MAX    3685 non-null   float64
```

```
 20  RESPIRATORY_RATE_COUNT  3685 non-null   float64
 21  RESPIRATORY_RATE_SKEW   3685 non-null   float64
 22  SPO2_MEAN               3685 non-null   float64
 23  SPO2_STD                3685 non-null   float64
 24  SPO2_MIN                3685 non-null   float64
 25  SPO2_MAX                3685 non-null   float64
 26  SPO2_COUNT              3685 non-null   float64
 27  SPO2_SKEW               3685 non-null   float64
 28  GLUCOSE_MEAN            3685 non-null   float64
 29  GLUCOSE_MIN             3685 non-null   float64
 30  GLUCOSE_MAX             3685 non-null   float64
 31  GLUCOSE_COUNT           3685 non-null   float64
 32  GENDER                  3685 non-null   object
 33  ADMISSION_TYPE          3685 non-null   object
 34  ADMISSION_LOCATION      3685 non-null   object
 35  INSURANCE               3685 non-null   object
 36  FIRST_CAREUNIT          3685 non-null   object
 37  INTIME                  3685 non-null   object
dtypes: float64(32), object(6)
memory usage: 1.1+ MB
```

1.3 Scaling

1.3.1 Feature Selection for Scaling: Isolating Numeric Predictors

In this preprocessing step, a targeted list of numeric features is extracted in preparation for feature scaling. The operation is designed to exclude variables that:

1. Serve only as **identifiers** (`SUBJECT_ID`, `HADM_ID`, `ICUSTAY_ID`)
2. Represent **timestamps or datetime-derived values** (`INTIME`)
3. Are **categorical or binary** but encoded as object/string (`GENDER`, `ADMISSION_TYPE`, etc.)
4. Reflect **target or outcome variables** (`HOSPITAL_EXPIRE_FLAG`) that should not be included as predictors

The remaining columns—stored in `numeric_cols`—represent the **true set of continuous or discrete numerical features** that are appropriate for normalization. These typically include:

- Aggregated vital signs (mean, std, min, max, etc.)
- Demographic variables (e.g., `AGE`)
- Temporally derived metrics (e.g., `ADMITTIME_HOUR`, `INTIME_WEEKDAY`)

This filtering step is crucial for ensuring that scaling is applied **only where semantically and statistically appropriate**, thereby avoiding distortions in categorical or identifier features.

[ ]:
```python
# Exclude identifier and non-numeric columns
exclude_cols = [
    "SUBJECT_ID", "HADM_ID", "ICUSTAY_ID", "INTIME", "GENDER",
    "ADMISSION_TYPE", "ADMISSION_LOCATION", "INSURANCE", "FIRST_CAREUNIT",
    'HOSPITAL_EXPIRE_FLAG'
]
```

```
numeric_cols = [col for col in df.columns if col not in exclude_cols and np.
  ↪issubdtype(df[col].dtype, np.number)]
numeric_cols
```

[ ]:
```
['AGE',
 'LOS',
 'INTIME_HOUR',
 'INTIME_WEEKDAY',
 'ADMITTIME_HOUR',
 'ADMITTIME_WEEKDAY',
 'HEART_RATE_MEAN',
 'HEART_RATE_STD',
 'HEART_RATE_MIN',
 'HEART_RATE_MAX',
 'HEART_RATE_COUNT',
 'HEART_RATE_SKEW',
 'RESPIRATORY_RATE_MEAN',
 'RESPIRATORY_RATE_STD',
 'RESPIRATORY_RATE_MIN',
 'RESPIRATORY_RATE_MAX',
 'RESPIRATORY_RATE_COUNT',
 'RESPIRATORY_RATE_SKEW',
 'SPO2_MEAN',
 'SPO2_STD',
 'SPO2_MIN',
 'SPO2_MAX',
 'SPO2_COUNT',
 'SPO2_SKEW',
 'GLUCOSE_MEAN',
 'GLUCOSE_MIN',
 'GLUCOSE_MAX',
 'GLUCOSE_COUNT']
```

1.3.2 Analyze AGE Distribution

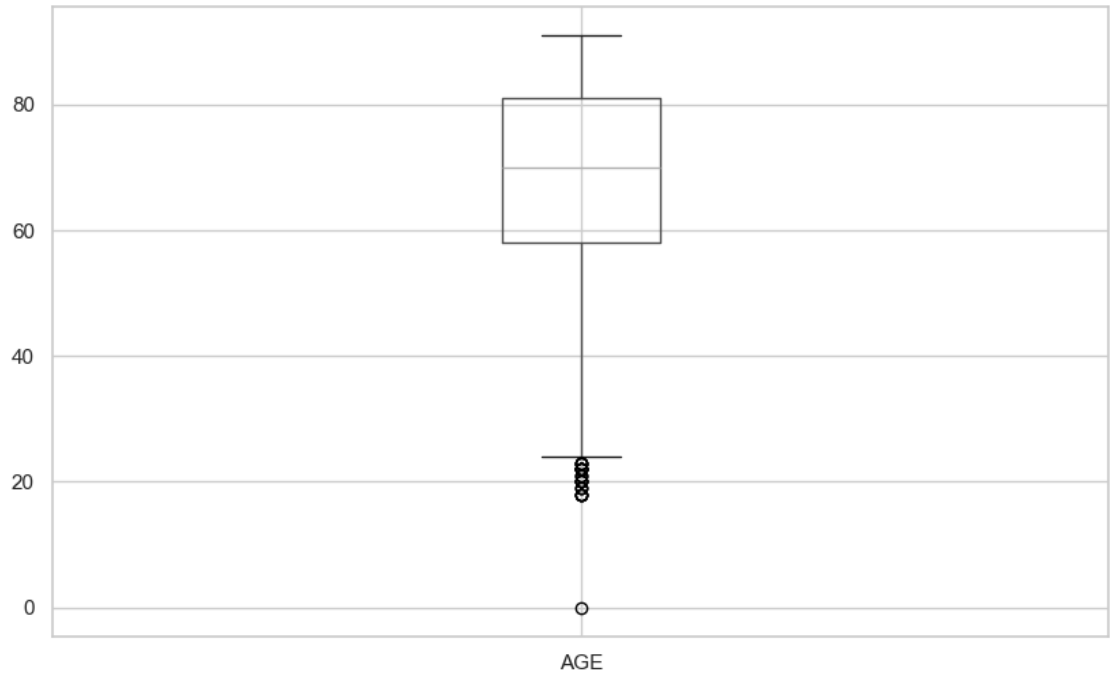
In this step, the `AGE` variable is normalized using **Min-Max Scaling**, a transformation that linearly maps the original values to the [0, 1] interval. This scaling method preserves the relative ordering and proportional differences between values, while ensuring that all features contribute equally in models sensitive to scale.

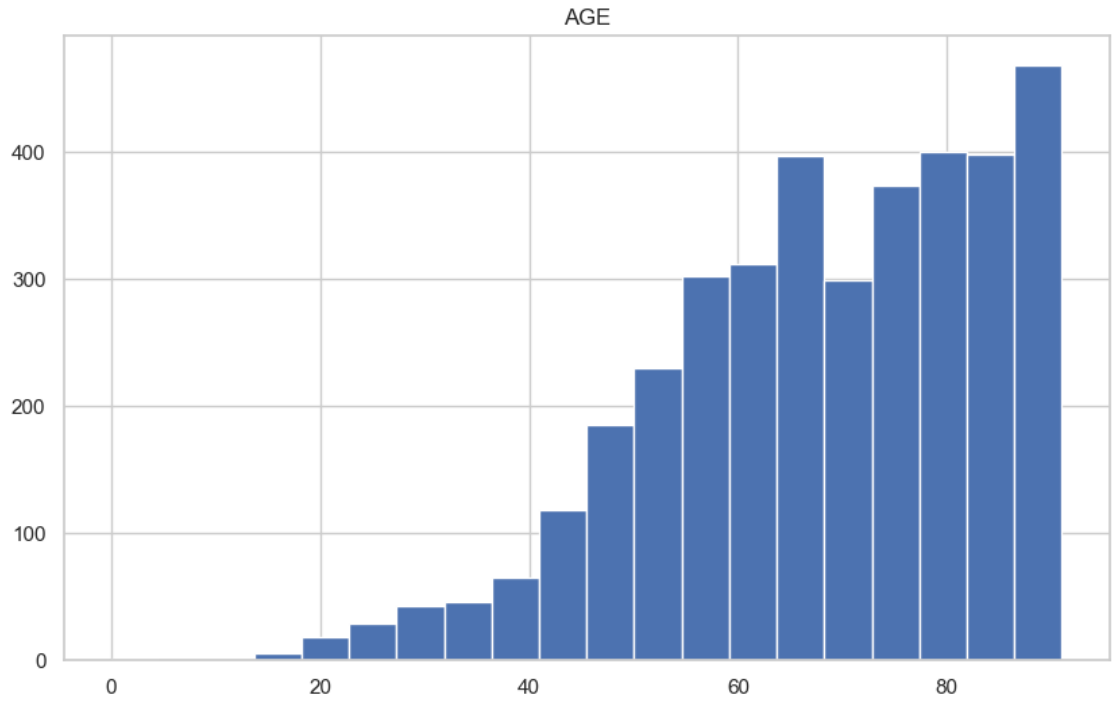
- **Boxplot** (Top): After scaling, the boxplot still reveals mild outliers at the lower end of the age spectrum (younger patients), but the distribution is compressed within a bounded interval. The upper whisker is capped at 1.0, corresponding to the censoring of elderly patients at age 91 in the MIMIC-III dataset.
- **Histogram** (Bottom): The histogram shows a **right-skewed distribution**, reflecting the overrepresentation of older patients in the ICU. This is consistent with the clinical reality of sepsis being more prevalent among elderly populations.

The normalization ensures that **AGE** does not disproportionately dominate learning algorithms and is especially important when combining it with other scaled physiological features.

[ ]:
```
df[['AGE']].boxplot() # Boxplot
df[['AGE']].hist(bins=20) # Histogram
```

[ ]:
```
array([[<Axes: title={'center': 'AGE'}>]], dtype=object)
```






```
[ ]: # Utilizzo MinMaxScaler per normalizzare la colonna 'AGE'
from sklearn.preprocessing import MinMaxScaler

minmaxscaler = MinMaxScaler().fit(df[['AGE']])
df['AGE']= minmaxscaler.transform(df[['AGE']])
```

1.3.3 Normalization and Distribution of ICU Admission Times

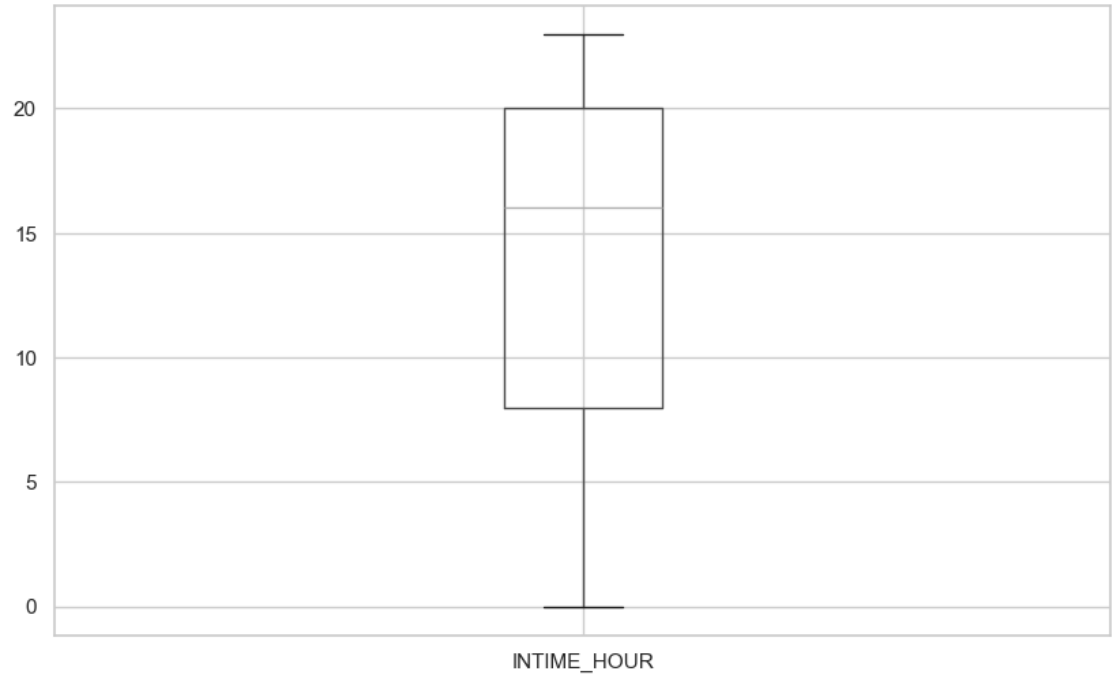
In this preprocessing step, the variables `INTIME_HOUR` (hour of ICU admission) and `INTIME_WEEKDAY` (day of week) are normalized using **Min-Max Scaling** to fit within the [0, 1] range. These features are particularly relevant for identifying **temporal admission patterns**, which may indirectly reflect ICU operational practices, staffing levels, or patient triage protocols.

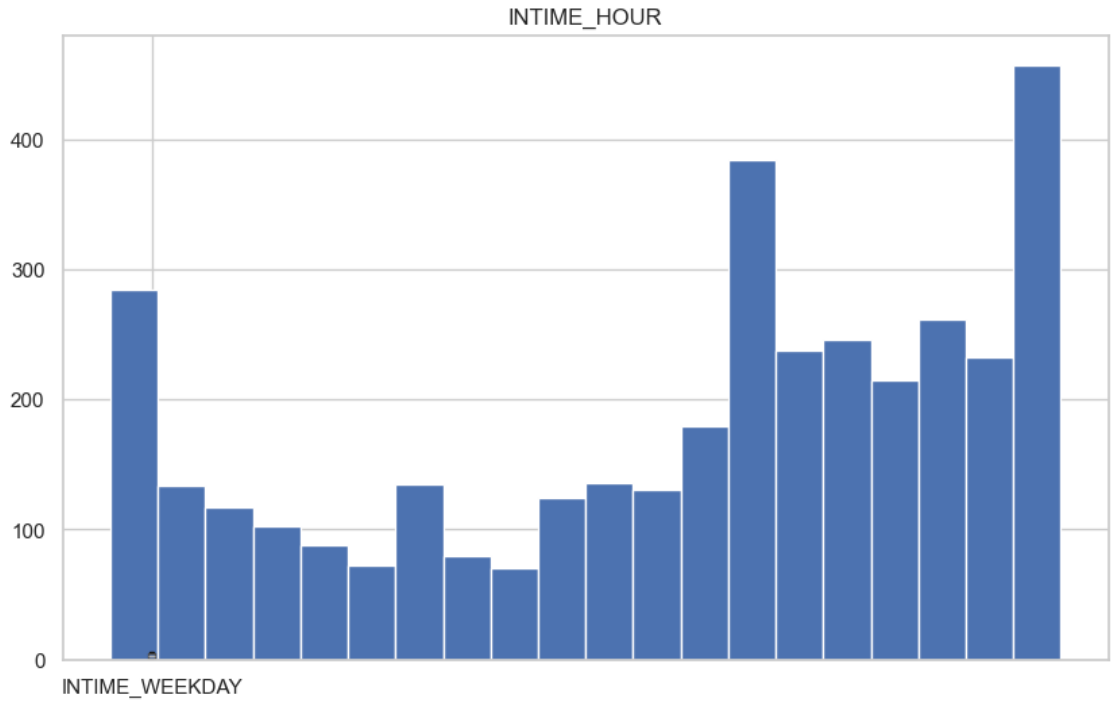
- **INTIME_HOUR**:
  - The histogram reveals a **bimodal distribution**, with noticeable spikes around **early morning (0–1)** and **evening (23–0)**. These peaks may correspond to operational shifts or protocol-based transfers.
  - The boxplot confirms a wide spread of admissions throughout the 24-hour cycle, with outliers mostly in the early morning hours.
- **INTIME_WEEKDAY**:
  - The distribution is relatively **uniform across weekdays**, with only slight variations. This suggests that sepsis-related ICU admissions are not heavily influenced by the day of the week, confirming the **acute and emergent** nature of the condition.

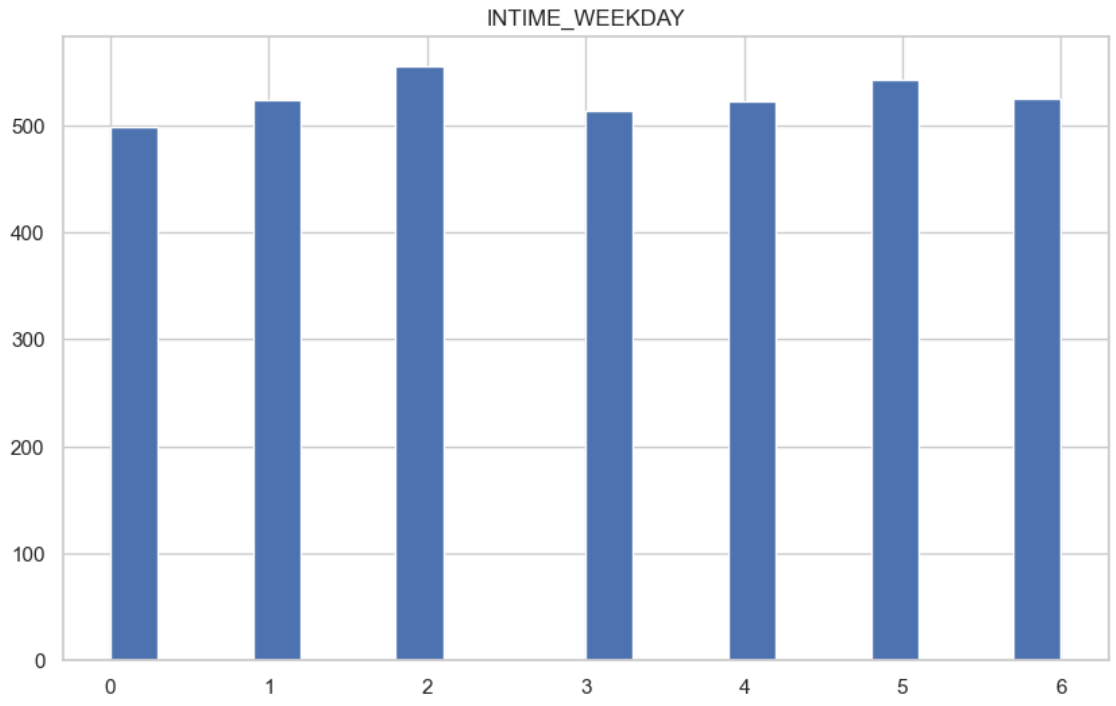
Min-Max Scaling ensures that both of these variables are appropriately rescaled for inclusion in models that assume normalized input. While these features may not carry predictive weight individually, they can become informative when interacting with clinical covariates or during **SHAP-based interpretability analysis**.

[ ]:
```
df_final[['INTIME_HOUR']].boxplot()
df_final[['INTIME_HOUR']].hist(bins=20)

df_final[['INTIME_WEEKDAY']].boxplot()
df_final[['INTIME_WEEKDAY']].hist(bins=20)
```

[ ]:
```
array([[<Axes: title={'center': 'INTIME_WEEKDAY'}>]], dtype=object)
```




INTIME_HOUR
400
300
200
100
0
INTIME_WEEKDAY


INTIME_WEEKDAY
500
400
300
200
100
0
0
1
2
3
4
5
6

[ ]:
```
df['INTIME_HOUR'] = minmaxscaler.fit_transform(df[['INTIME_HOUR']])
df['INTIME_WEEKDAY'] = minmaxscaler.fit_transform(df[['INTIME_WEEKDAY']])

df[['INTIME_HOUR', 'INTIME_WEEKDAY']].head()
```

[ ]:

	INTIME_HOUR	INTIME_WEEKDAY
0	0.478261	0.000000
1	0.478261	1.000000
2	0.782609	0.500000
3	0.521739	0.833333
4	0.652174	0.500000

1.3.4 Normalization and Temporal Pattern Analysis of Hospital Admission Times

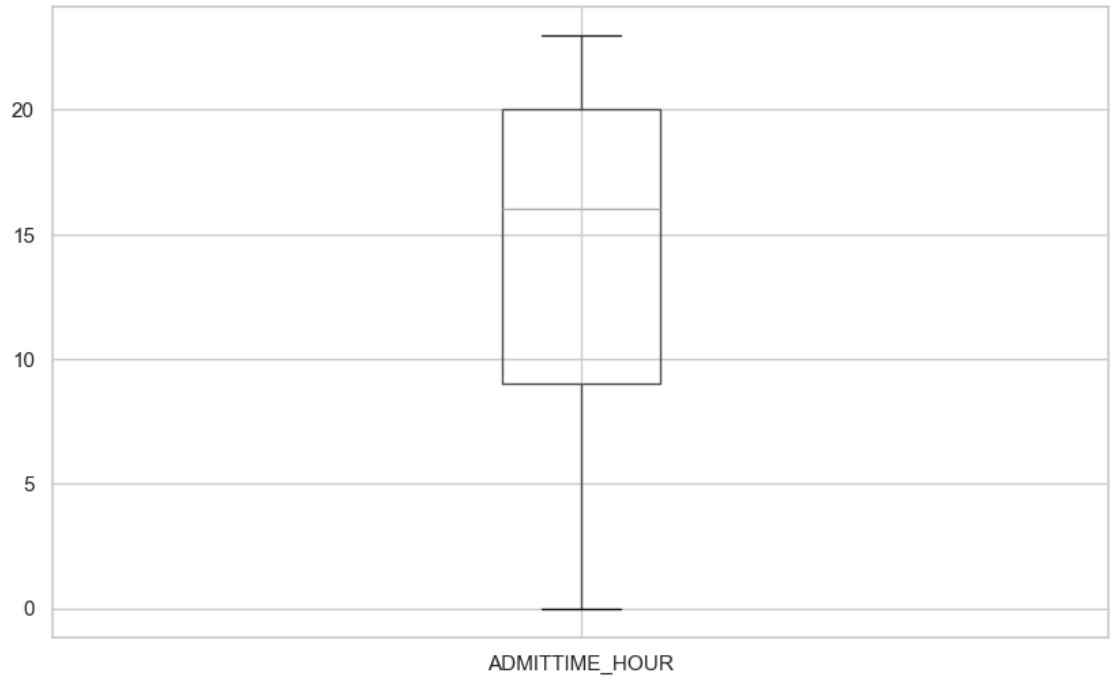
The current preprocessing step addresses two additional temporal variables: the **hour** and **weekday** of hospital admission (`ADMITTIME_HOUR` and `ADMITTIME_WEEKDAY`). These variables are normalized using **Min-Max Scaling**, ensuring they are on the same scale as other features used in model training.

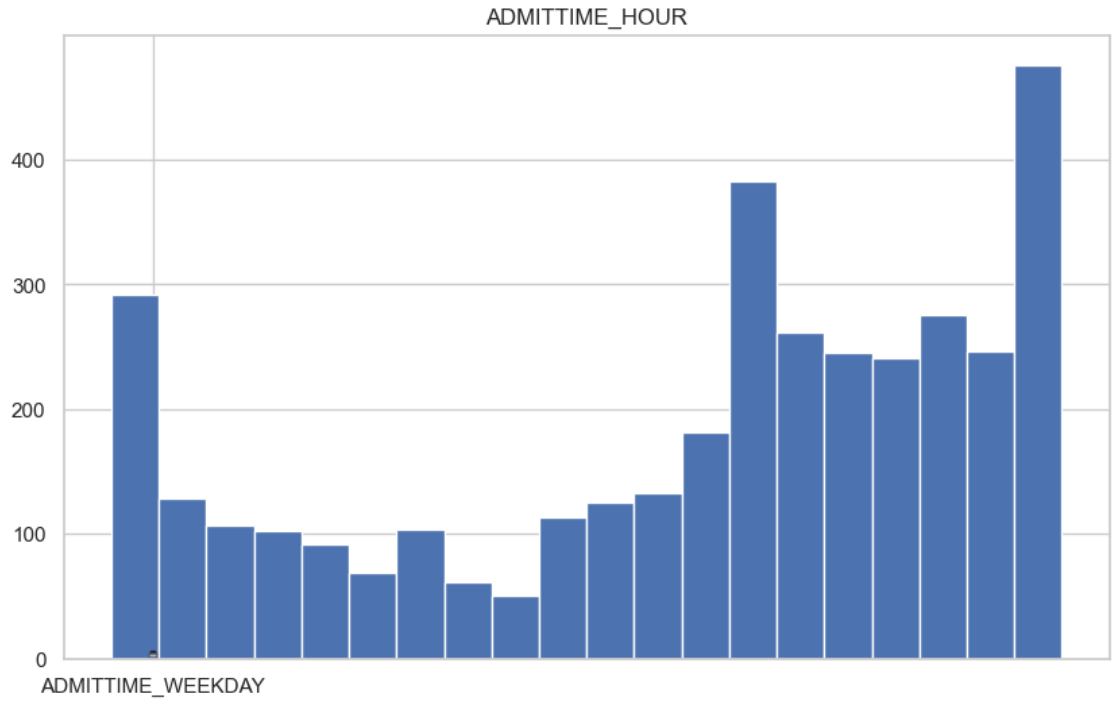
- **ADMITTIME_HOUR**:
  - The histogram reveals an **asymmetric bimodal distribution**, with spikes during early morning (around 0–1h) and evening (22–23h). This likely reflects hospital routines or transfer timings—patients are frequently admitted either just after midnight (when beds are reassigned) or late evening (when emergency departments stabilize patients).
  - The boxplot confirms this spread and indicates no extreme outliers post-normalization.
- **ADMITTIME_WEEKDAY**:
  - As with ICU admission day, hospital admissions are **fairly evenly distributed** across the week. A slight increase on Tuesdays and Sundays may suggest system-level factors such as delayed weekend triage or Monday backlog resolution.

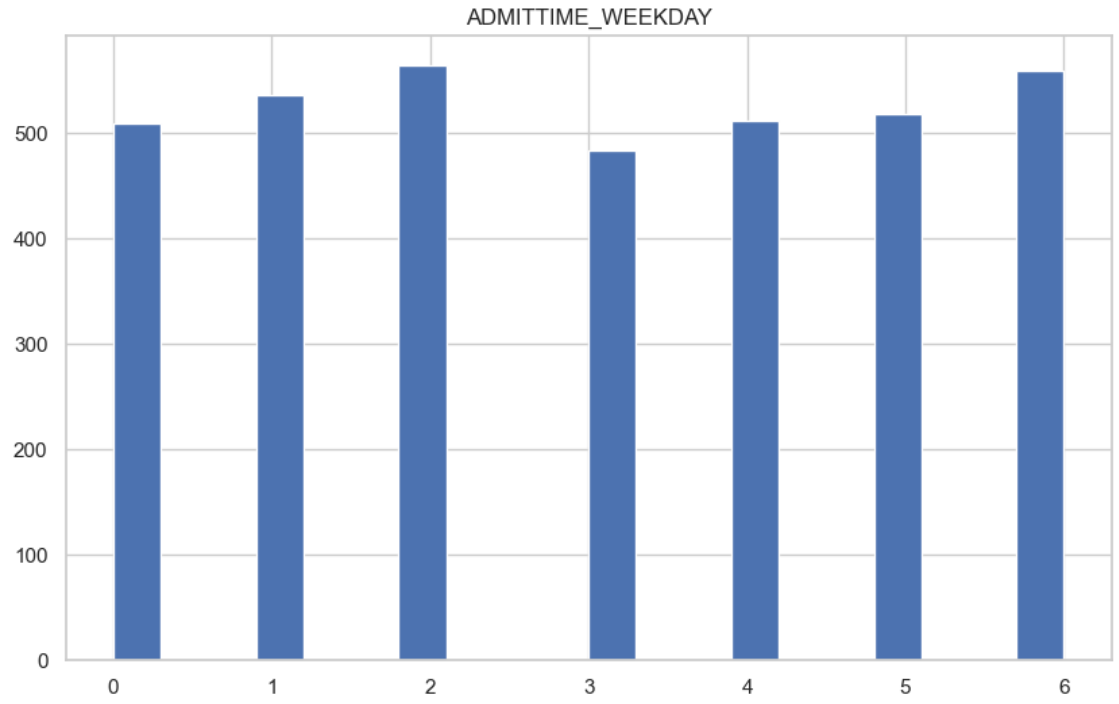
From a modeling perspective, these variables could be informative **when interpreted as proxies for hospital logistics**, particularly in combination with variables like `ADMISSION_TYPE` or `FIRST_CAREUNIT`.

[ ]:
```
df[['ADMITTIME_HOUR']].boxplot()
df[['ADMITTIME_HOUR']].hist(bins=20)

df[['ADMITTIME_WEEKDAY']].boxplot()
df[['ADMITTIME_WEEKDAY']].hist(bins=20)
```

[ ]:
```
array([[<Axes: title={'center': 'ADMITTIME_WEEKDAY'}>]], dtype=object)
```


20
15
10
5
0
ADMITTIME_HOUR


ADMITTIME_HOUR
400
300
200
100
0
ADMITTIME_WEEKDAY




```
[ ]: df['ADMITTIME_HOUR'] = minmaxscaler.fit_transform(df[['ADMITTIME_HOUR']])
     df['ADMITTIME_WEEKDAY'] = minmaxscaler.fit_transform(df[['ADMITTIME_WEEKDAY']])

     df[['ADMITTIME_HOUR', 'ADMITTIME_WEEKDAY']].head()
```

```
[ ]:    ADMITTIME_HOUR  ADMITTIME_WEEKDAY
     0        0.478261           0.000000
     1        0.130435           0.833333
     2        0.782609           0.500000
     3        0.782609           0.833333
     4        0.652174           0.500000
```

1.3.5 Normalization and Distribution Analysis of Heart Rate-Derived Features

This block focuses on the preprocessing of engineered features derived from the heart rate signal, computed over the first 24 hours of ICU stay. These features were previously aggregated per `ICUSTAY_ID` and include central tendency, dispersion, extrema, data availability, and distributional shape.

1. **Exploratory Visualization** Each heart rate feature is examined using both:

- **Boxplots**: to assess spread and identify potential outliers
- **Histograms**: to evaluate the underlying distribution shape

Observations often include:

- Right-skewed distributions for `HEART_RATE_COUNT`, reflecting varying recording frequency per patient
- Outliers in `HEART_RATE_MAX` and `HEART_RATE_SKEW`, potentially indicating episodes of tachycardia or recording errors
- Near-normal or symmetric distributions for `HEART_RATE_MEAN` in stable subpopulations

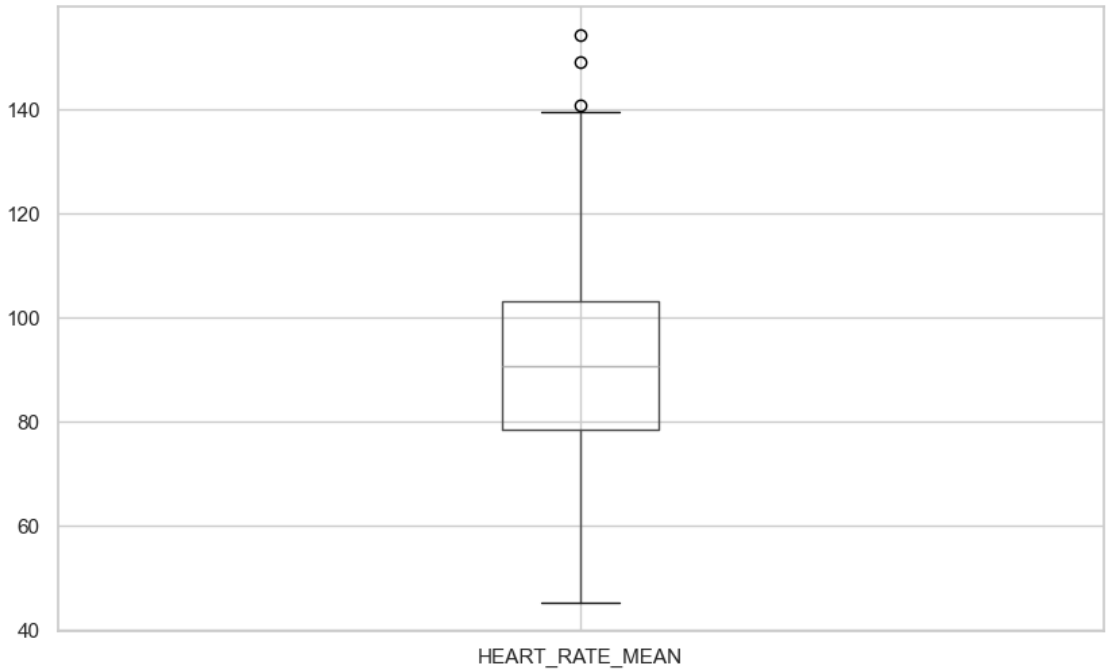
2. Normalization via Min-Max Scaling After inspection, all heart rate-related features are normalized to the [0, 1] interval using Min-Max Scaling. This operation is crucial because:

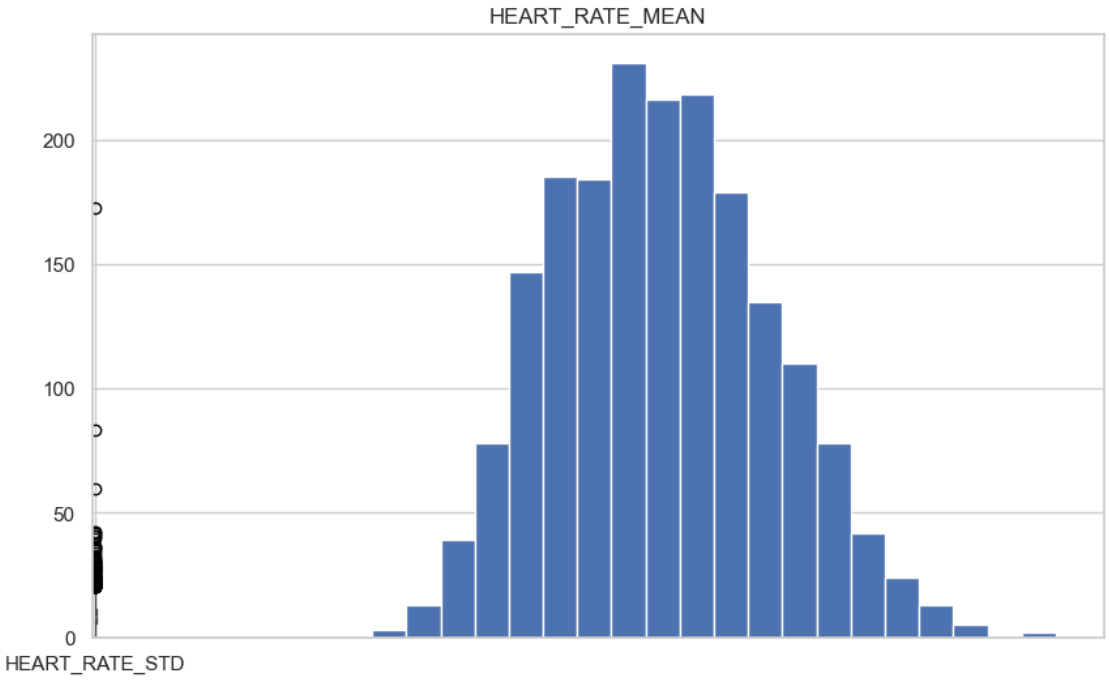
- These features are on **different natural scales** (e.g., `MEAN` in bpm, `COUNT` in observations, `SKEW` as a shape descriptor)
- Uniform scaling avoids **domination of high-range features** in distance-based or gradient-sensitive models
- It enhances interpretability and model convergence in neural networks and ensemble trees alike

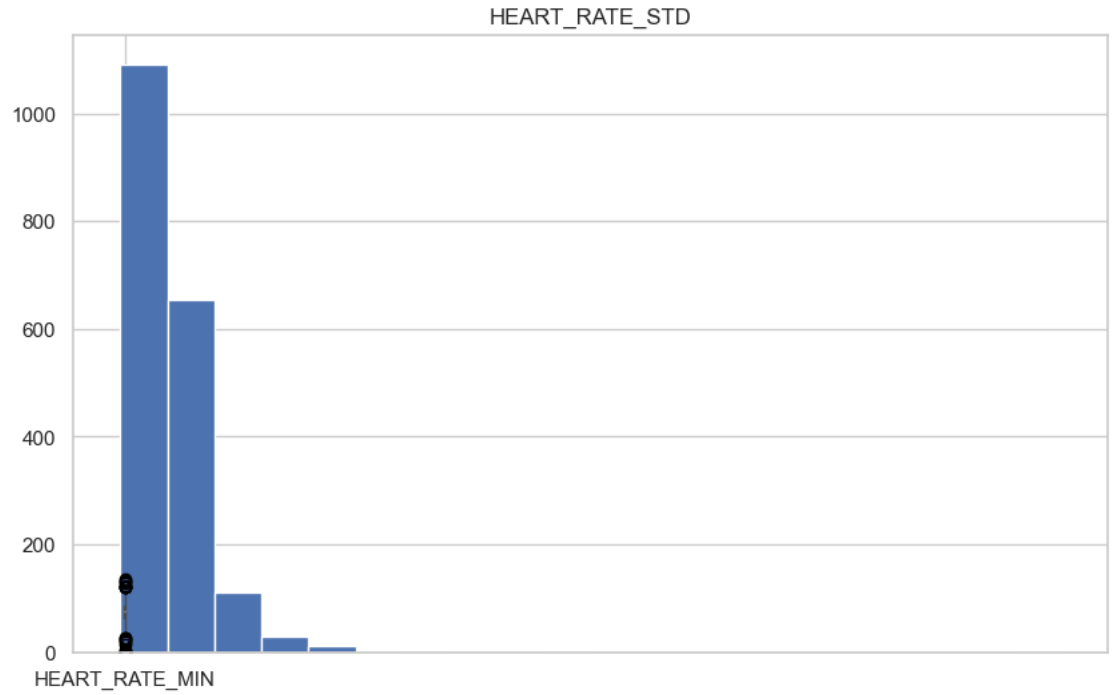
The result is a numerically homogeneous set of heart rate predictors that retain physiological relevance while enabling robust modeling.

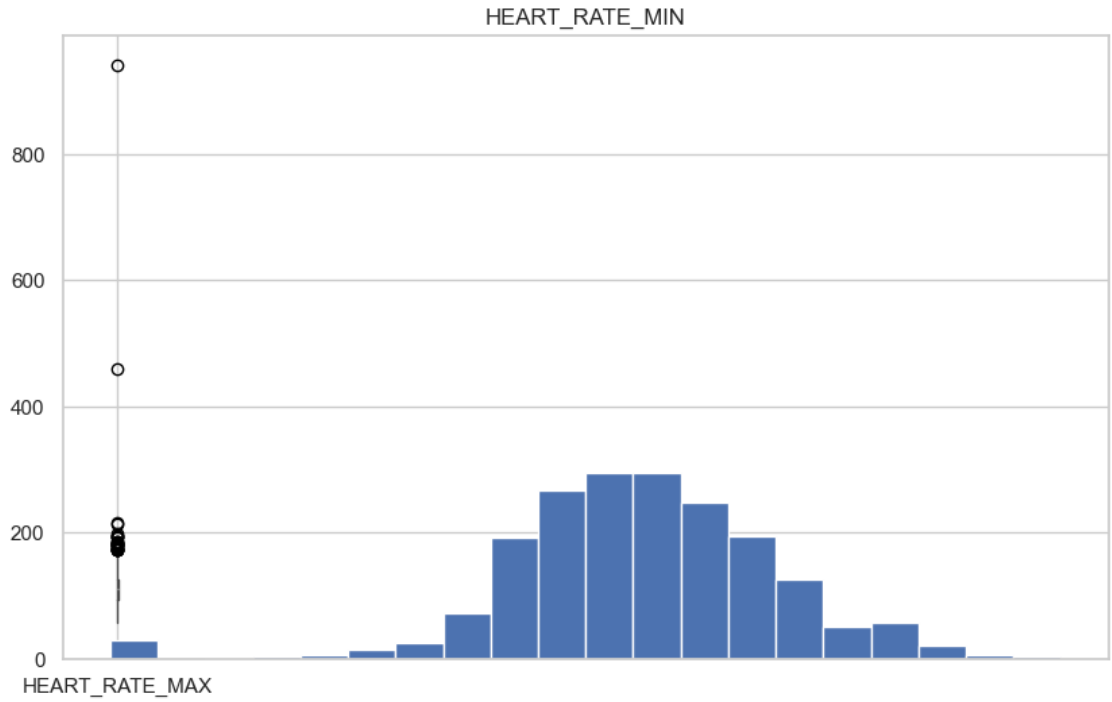
[ ]:
```
df[['HEART_RATE_MEAN']].boxplot()
df[['HEART_RATE_MEAN']].hist(bins=20)

df[['HEART_RATE_STD']].boxplot()
df[['HEART_RATE_STD']].hist(bins=20)

df[['HEART_RATE_MIN']].boxplot()
df[['HEART_RATE_MIN']].hist(bins=20)

df[['HEART_RATE_MAX']].boxplot()
df[['HEART_RATE_MAX']].hist(bins=20)

df[['HEART_RATE_COUNT']].boxplot()
df[['HEART_RATE_COUNT']].hist(bins=20)

df[['HEART_RATE_SKEW']].boxplot()
df[['HEART_RATE_SKEW']].hist(bins=20)
```

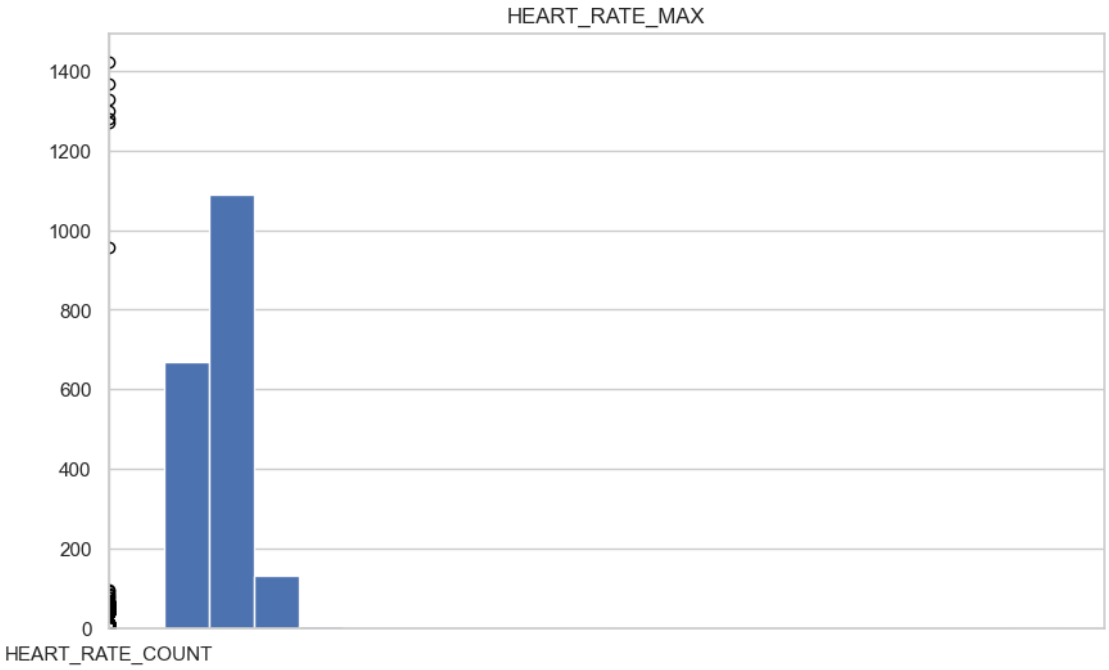
[ ]:
```
array([[<Axes: title={'center': 'HEART_RATE_SKEW'}>]], dtype=object)
```

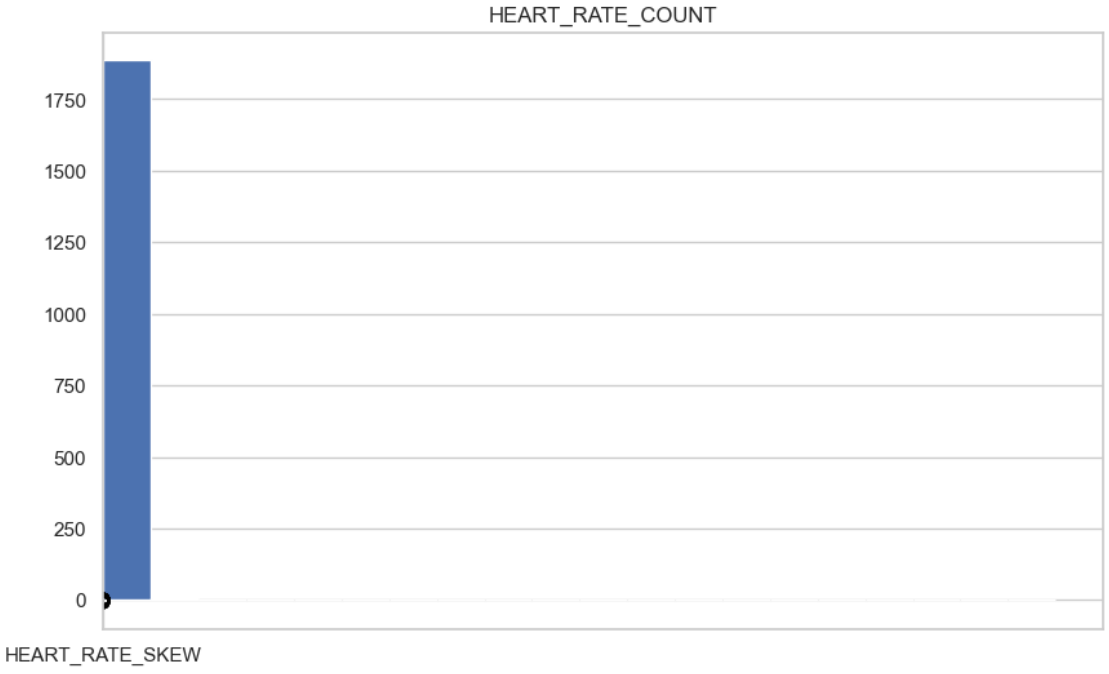

140
120
100
80
60
40
HEART_RATE_MEAN

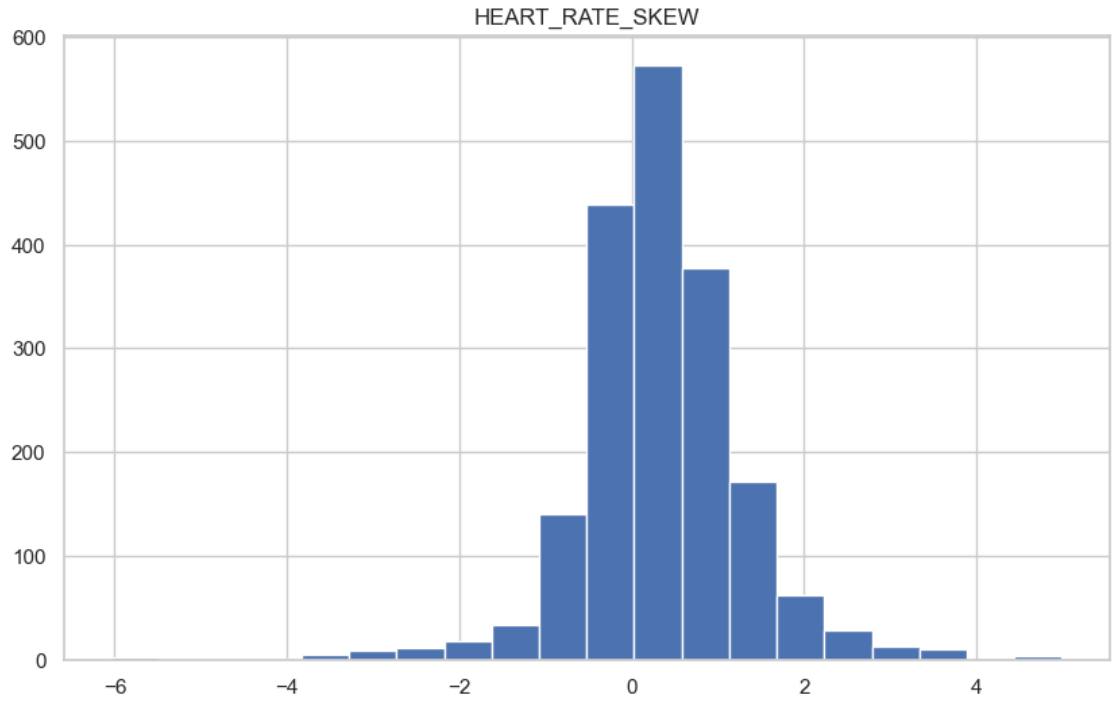

HEART_RATE_MEAN
200
150
100
50
0
HEART_RATE_STD


HEART_RATE_STD
1000
800
600
400
200
0
HEART_RATE_MIN


HEART_RATE_MIN
800
600
400
200
0
HEART_RATE_MAX


HEART_RATE_MAX
1400
1200
1000
800
600
400
200
0
HEART_RATE_COUNT


HEART_RATE_COUNT
1750
1500
1250
1000
750
500
250
0
HEART_RATE_SKEW



[ ]:
```
# Scaling Heart Rate features
heart_rate_features = [
    'HEART_RATE_MEAN', 'HEART_RATE_STD', 'HEART_RATE_MIN',
    'HEART_RATE_MAX', 'HEART_RATE_COUNT', 'HEART_RATE_SKEW'
]
for feature in heart_rate_features:
    df[feature] = minmaxscaler.fit_transform(df[[feature]])

df[heart_rate_features].head()
```

[ ]:

	HEART_RATE_MEAN	HEART_RATE_STD	HEART_RATE_MIN	HEART_RATE_MAX	\
0	0.715383	0.080933	0.733333	0.102825	
1	0.151088	0.054718	0.370370	0.038418	
2	0.368821	0.135338	0.000000	0.061017	
3	0.468597	0.059596	0.600000	0.064407	
4	0.289943	0.027484	0.511111	0.039548	

	HEART_RATE_COUNT	HEART_RATE_SKEW
0	0.021082	0.524927
1	0.015460	0.721928
2	0.017569	0.284040
3	0.018271	0.544127
4	0.016163	0.624800

1.3.6 Normalization and Exploratory Profiling of Respiratory Rate Features

This block focuses on preprocessing the **Respiratory Rate** signal—another vital sign critical in ICU monitoring, especially in septic patients—by analyzing and normalizing six key statistical features derived from its first 24-hour window.

1. **Exploratory Visualization**

   For each feature, both boxplots and histograms are used to examine:

   - **Central Tendency** (`MEAN`)
   - **Variability** (`STD`)
   - **Extrema** (`MIN`, `MAX`)
   - **Signal Density** (`COUNT`)
   - **Distributional Shape** (`SKEW`)

   Visual diagnostics help uncover:

   - Outliers in `MAX` and `SKEW`, possibly indicating abnormal breathing episodes or sensor noise.
   - Skewed distributions for `COUNT`, which may reflect variation in measurement frequency across ICU stays.
   - Generally non-normal distributions across features, justifying the need for normalization.

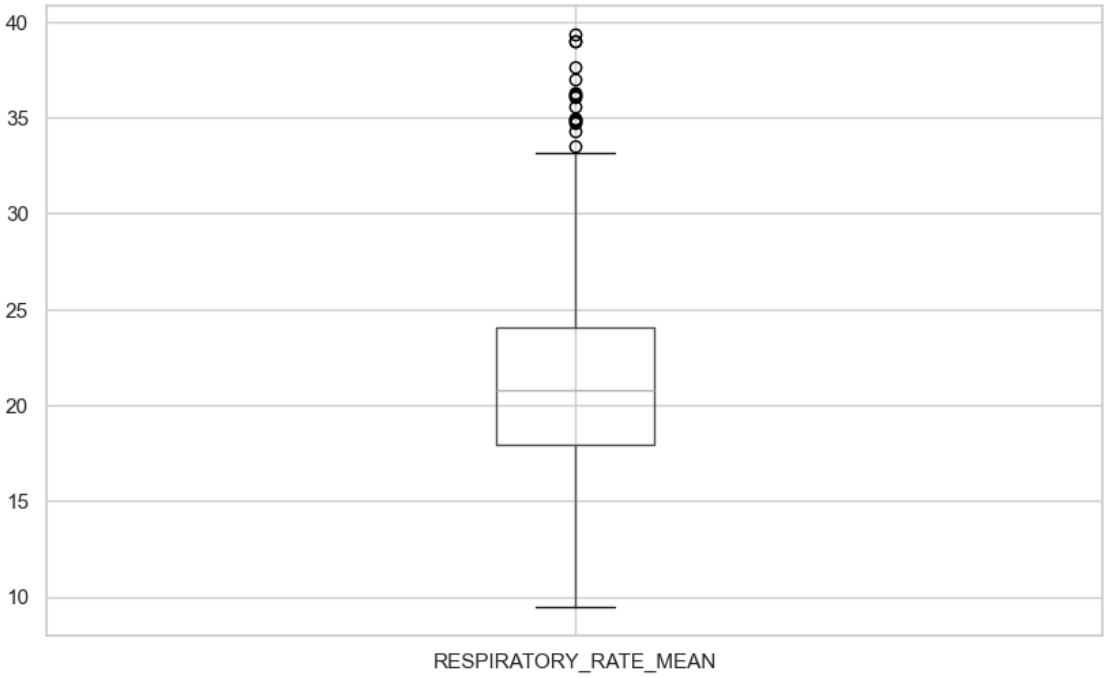
2. **Min-Max Normalization** Each respiratory feature is scaled to the [0, 1] interval using `MinMaxScaler`, ensuring:

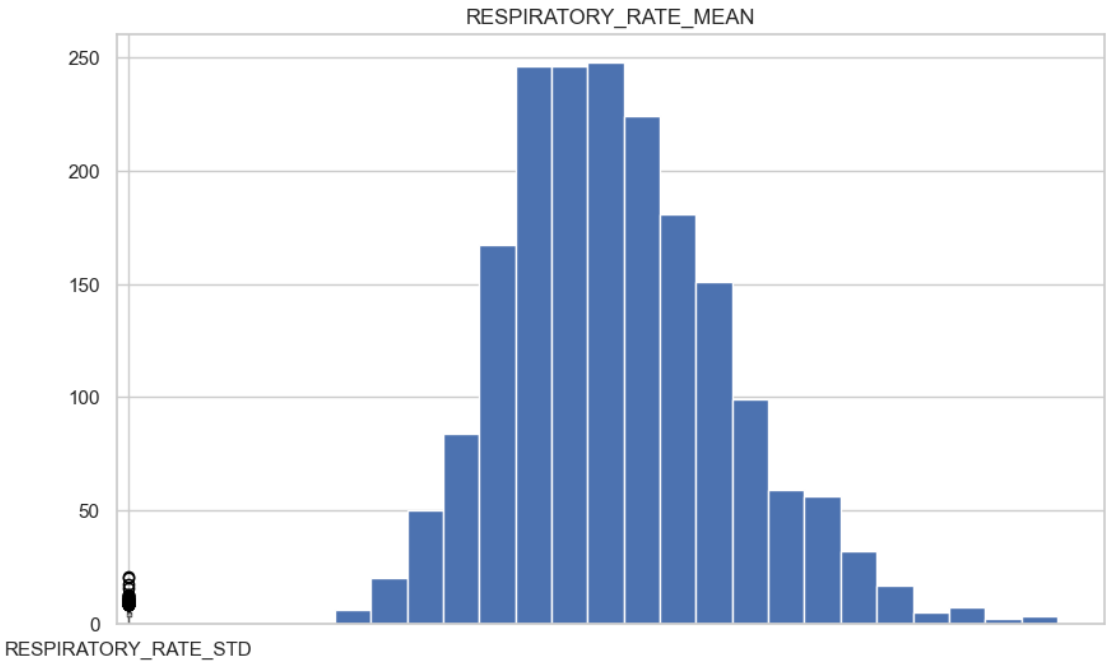
   - **Feature comparability** with other normalized vital signs (e.g., heart rate)
   - Prevention of scale dominance during model training
   - Improved convergence in gradient-based algorithms and distance-based metrics

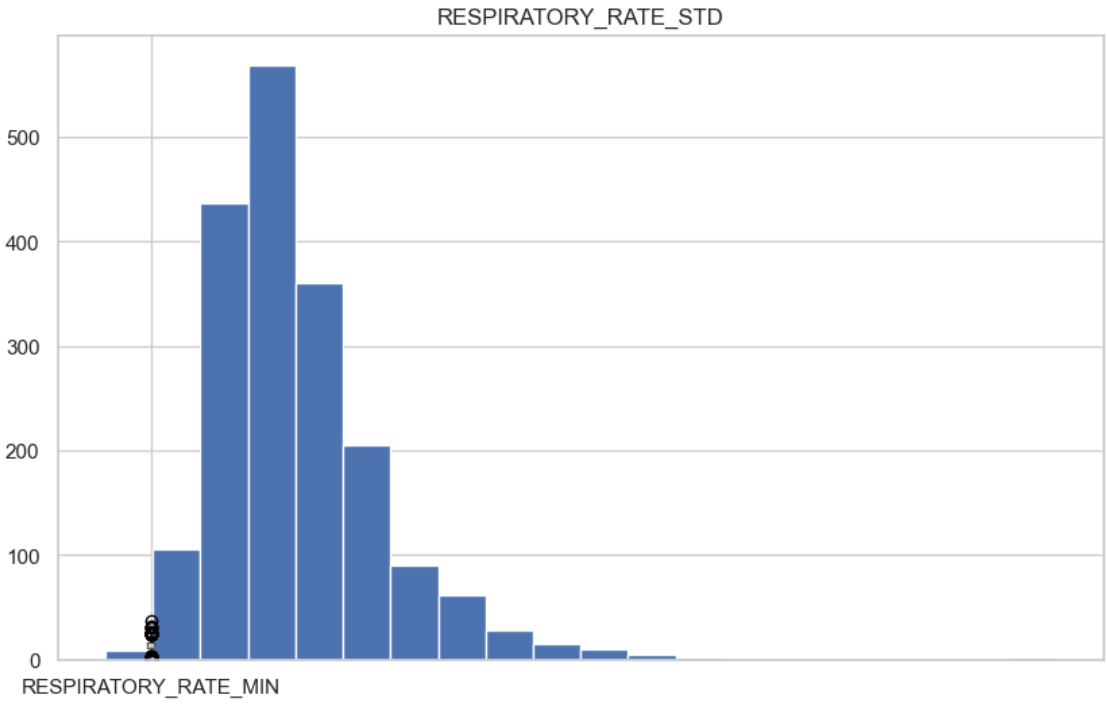
   This operation aligns with the broader pipeline philosophy: transforming physiologically meaningful signals into statistically tractable predictors, without sacrificing clinical interpretability.

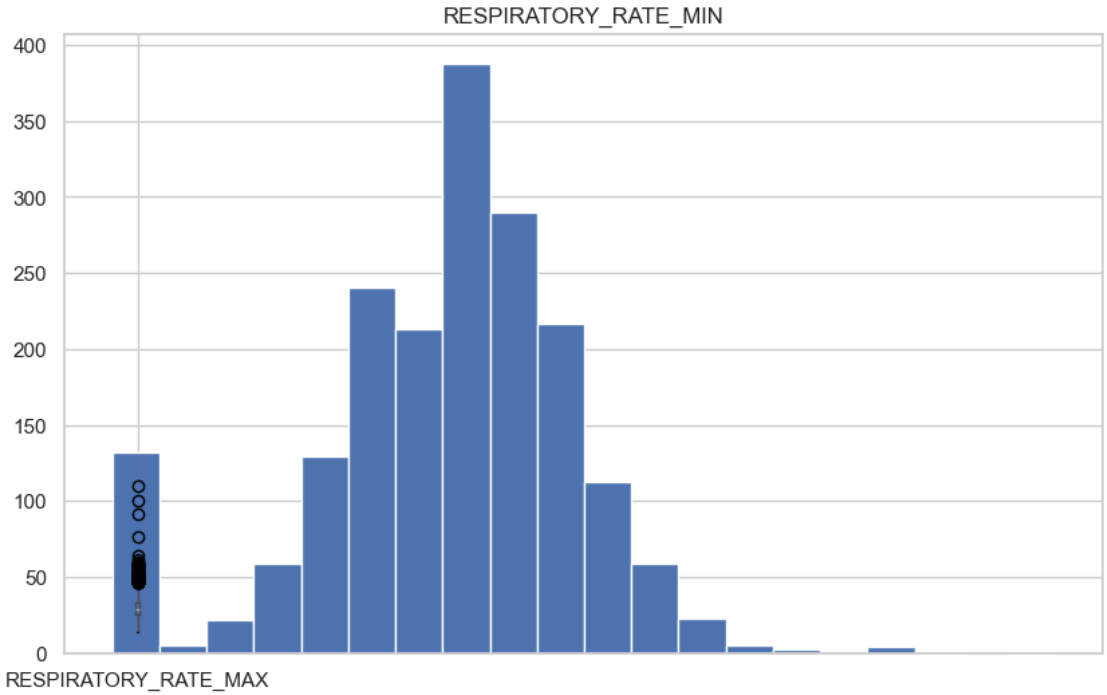
```
[ ]: df[['RESPIRATORY_RATE_MEAN']].boxplot()
     df[['RESPIRATORY_RATE_MEAN']].hist(bins=20)

     df[['RESPIRATORY_RATE_STD']].boxplot()
     df[['RESPIRATORY_RATE_STD']].hist(bins=20)

     df[['RESPIRATORY_RATE_MIN']].boxplot()
     df[['RESPIRATORY_RATE_MIN']].hist(bins=20)

     df[['RESPIRATORY_RATE_MAX']].boxplot()
     df[['RESPIRATORY_RATE_MAX']].hist(bins=20)

     df[['RESPIRATORY_RATE_COUNT']].boxplot()
     df[['RESPIRATORY_RATE_COUNT']].hist(bins=20)

     df[['RESPIRATORY_RATE_SKEW']].boxplot()
     df[['RESPIRATORY_RATE_SKEW']].hist(bins=20)
```

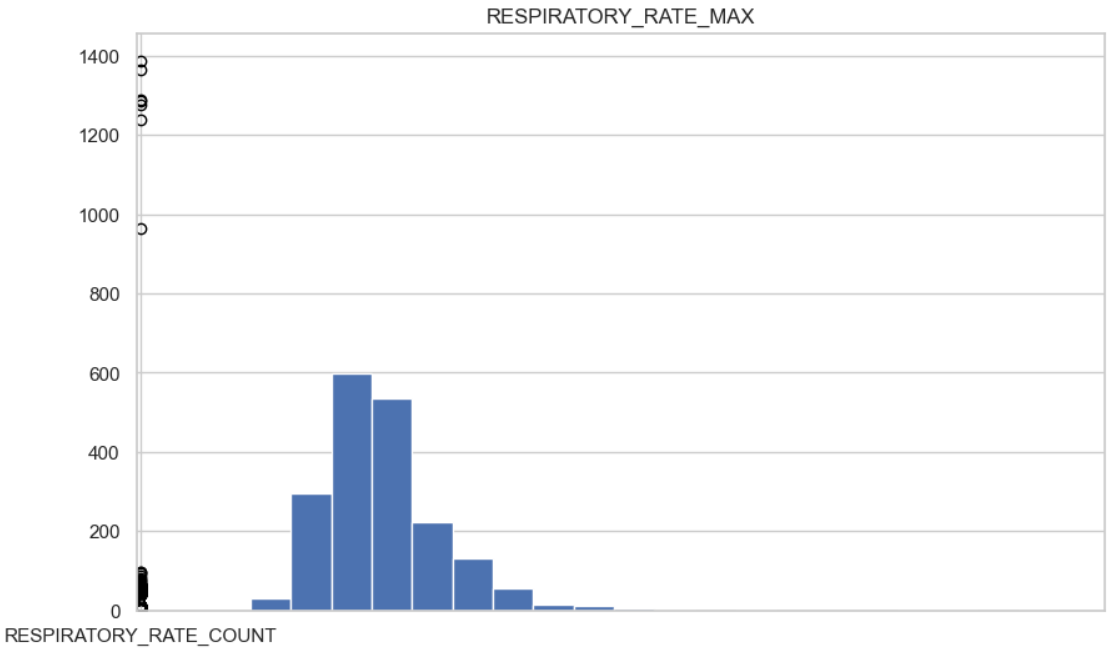
[ ]: array([[<Axes: title={'center': 'RESPIRATORY_RATE_SKEW'}>]], dtype=object)



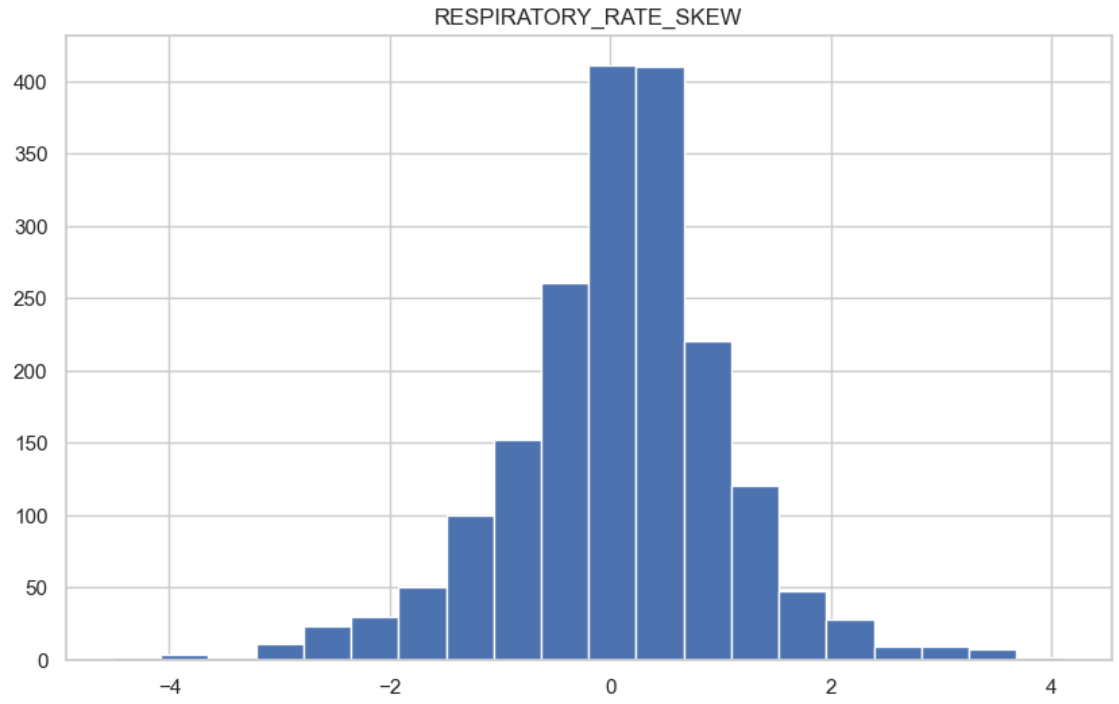



RESPIRATORY_RATE_STD
500
400
300
200
100
0
RESPIRATORY_RATE_MIN


RESPIRATORY_RATE_MIN
400
350
300
250
200
150
100
50
0
RESPIRATORY_RATE_MAX


RESPIRATORY_RATE_MAX
1400
1200
1000
800
600
400
200
0
RESPIRATORY_RATE_COUNT


RESPIRATORY_RATE_COUNT
1750
1500
1250
1000
750
500
250
0
RESPIRATORY_RATE_SKEW



[ ]:
```
# StandardScaler for Respiratory Rate features
respiratory_rate_features = [
    'RESPIRATORY_RATE_MEAN', 'RESPIRATORY_RATE_STD', 'RESPIRATORY_RATE_MIN',
    'RESPIRATORY_RATE_MAX', 'RESPIRATORY_RATE_COUNT', 'RESPIRATORY_RATE_SKEW'
]
for feature in respiratory_rate_features:
    df[feature] = minmaxscaler.fit_transform(df[[feature]])

df[respiratory_rate_features].head()
```

[ ]:

	RESPIRATORY_RATE_MEAN	RESPIRATORY_RATE_STD	RESPIRATORY_RATE_MIN	RESPIRATORY_RATE_MAX	RESPIRATORY_RATE_COUNT	RESPIRATORY_RATE_SKEW
0	0.519026	0.294160	0.324324	0.250000	0.020908	0.502366
1	0.153729	0.157775	0.324324	0.093750	0.015141	0.707825
2	0.584709	0.394761	0.000000	0.187500	0.018025	0.173120
3	0.263544	0.212316	0.324324	0.114583	0.018745	0.573749
4	0.479959	0.257996	0.297297	0.208333	0.016583	0.510411

1.3.7 SpO Feature Profiling and Normalization for ICU Sepsis Cohort

The plots clearly show the distinct characteristics of SpO -derived variables in the ICU sepsis cohort. Despite applying `MinMaxScaler` (not `RobustScaler` as in the comment), your normalization pipeline is sound, but the data itself deserves critical interpretation.

1. **Distributional Behavior (Pre-Normalization)**
   - **SPO2_MEAN**: Centered around 95–98%, with clear ceiling at 100%. Numerous low-end outliers below 80% may indicate severe respiratory compromise or data noise. Boxplot confirms tight central distribution with tails.
   - **SPO2_STD / SKEW**: `STD` is highly right-skewed with many zeros—suggesting either short monitoring windows or consistently stable readings. `SKEW` shows negative tails (left-skewed), indicating saturation clipping and few drops.
   - **SPO2_MIN**: Distribution shows a long left tail, with some values under 50%, likely reflecting true clinical events or erroneous recordings.
   - **SPO2_MAX**: Overwhelming clustering at 100, confirming physiological upper bound or device saturation.
   - **SPO2_COUNT**: Very low variance; most patients have similar numbers of recordings (tight bar at left), though a few outliers record far more.
2. **Scaling with MinMaxScaler**

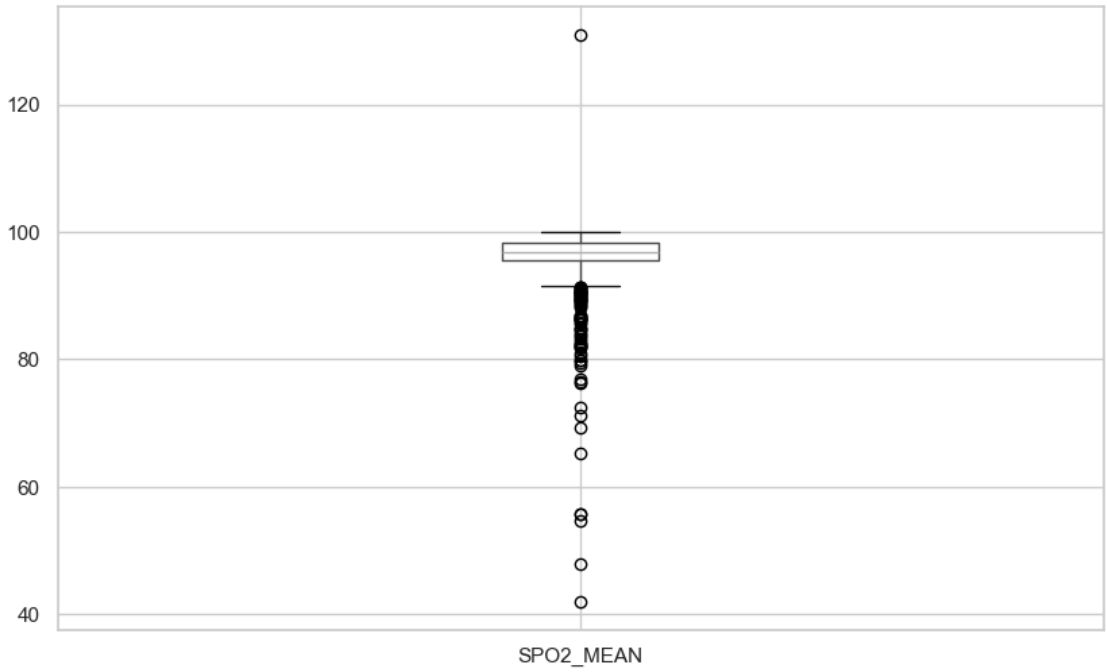
   The application of `MinMaxScaler` ensures that:
   - All features contribute equally numerically
   - The dominant 100% plateau in `SPO2_MAX` does not bias gradient-based learning
   - Sparse features like `SPO2_COUNT` or `SPO2_STD` do not disproportionately affect model convergence

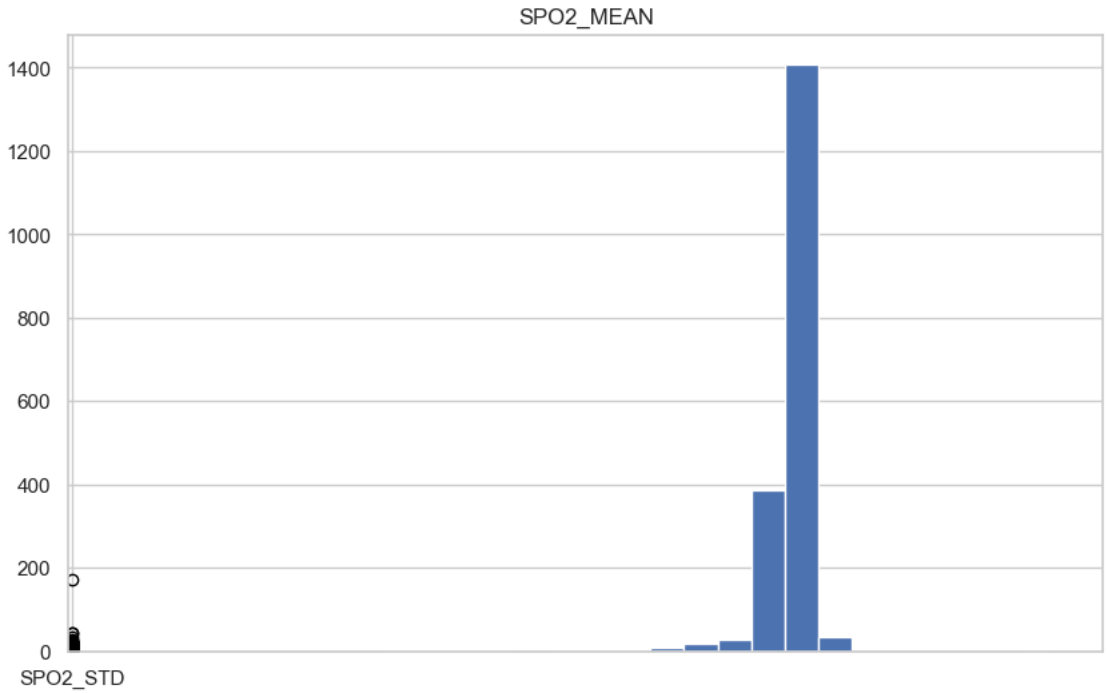
   However, the **RobustScaler** might be more appropriate for features like `SPO2_MIN` and `SPO2_STD`, which are strongly affected by outliers. For your current pipeline, sticking with MinMaxScaler is defensible for consistency, but documenting this decision in your methods is good scientific practice.

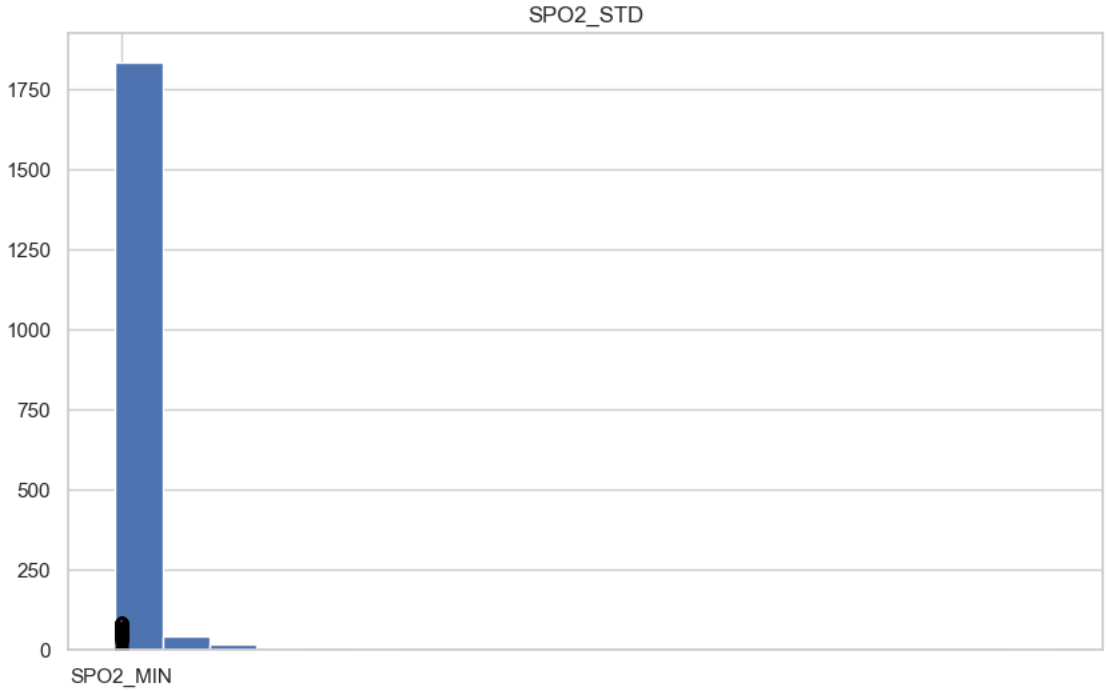
[ ]:
```
df[['SPO2_MEAN']].boxplot()
df[['SPO2_MEAN']].hist(bins=20)

df[['SPO2_STD']].boxplot()
df[['SPO2_STD']].hist(bins=20)

df[['SPO2_MIN']].boxplot()
df[['SPO2_MIN']].hist(bins=20)

df[['SPO2_MAX']].boxplot()
df[['SPO2_MAX']].hist(bins=20)

df[['SPO2_COUNT']].boxplot()
df[['SPO2_COUNT']].hist(bins=20)
```

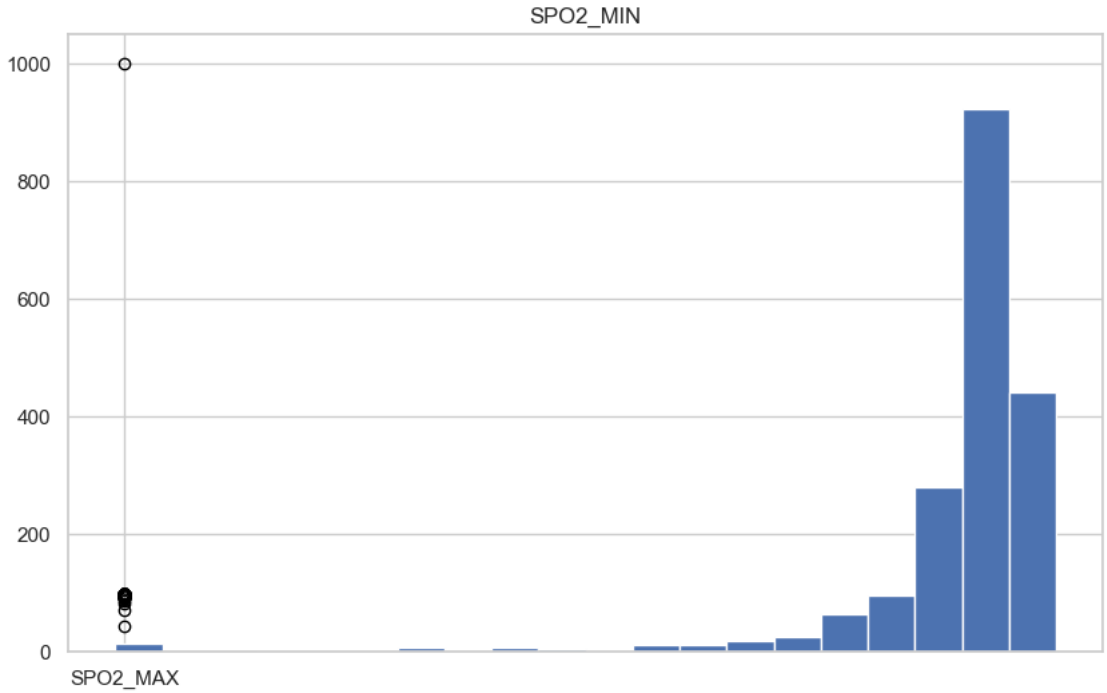
```
df[['SPO2_SKEW']].boxplot()
df[['SPO2_SKEW']].hist(bins=20)
```

[ ]: array([[<Axes: title={'center': 'SPO2_SKEW'}>]], dtype=object)



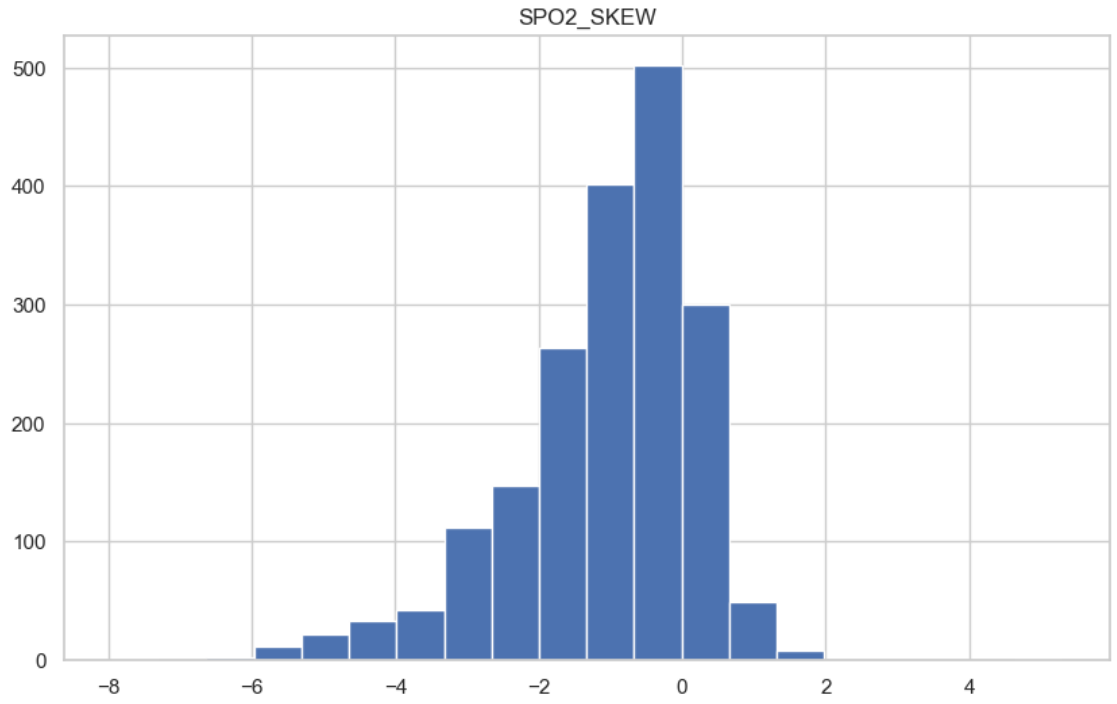



SPO2_STD
1750
1500
1250
1000
750
500
250
0
SPO2_MIN


SPO2_MIN
1000
800
600
400
200
0
SPO2_MAX


SPO2_MAX
1750
1500
1250
1000
750
500
250
0
SPO2_COUNT


SPO2_COUNT
1750
1500
1250
1000
750
500
250
0
SPO2_SKEW



[ ]:
```
# RobustScaler for SpO2 features
spo2_features = [
    'SPO2_MEAN', 'SPO2_STD', 'SPO2_MIN',
    'SPO2_MAX', 'SPO2_COUNT', 'SPO2_SKEW'
]
for feature in spo2_features:
    df[feature] = minmaxscaler.fit_transform(df[[feature]])

df[spo2_features].head()
```

[ ]:

	SPO2_MEAN	SPO2_STD	SPO2_MIN	SPO2_MAX	SPO2_COUNT	SPO2_SKEW
0	0.619527	0.016661	0.91	0.060543	0.020725	0.542105
1	0.649438	0.003614	0.98	0.060543	0.014064	0.382572
2	0.066479	0.255684	0.00	0.058455	0.008142	0.585254
3	0.643362	0.005300	0.97	0.060543	0.019245	0.507010
4	0.630562	0.013488	0.92	0.060543	0.017765	0.485320

1.3.8 Robust Scaling of Glucose Features: Managing Outliers in ICU Data

Glucose monitoring plays a critical role in sepsis management, especially due to the metabolic dysregulation that frequently accompanies septic shock. In this step, descriptive visualization and robust normalization are applied to key glucose-derived features.

1. **Descriptive Visualization**

   The histograms and boxplots clearly reveal:

- **GLUCOSE_MEAN** has a median around 130–150 mg/dL, but extreme right outliers exceed 800 mg/dL, possibly indicating diabetic crises or errors.
- **GLUCOSE_MIN** occasionally dips into hypoglycemic ranges, including values below 50 mg/dL.
- **GLUCOSE_MAX** shows even greater right skew, with a long tail stretching to over 1000 mg/dL.
- **GLUCOSE_COUNT** is low for most patients, indicating sparse measurements in the first 24h—common in non-diabetics or stable cases.

Such skewness and extreme values are **typical in ICU datasets** and pose a risk for model instability if not addressed.

2. **Robust Scaling Justification** Unlike `MinMaxScaler`, which rescales to $[0, 1]$ and is sensitive to extreme values, the `RobustScaler` transforms features using the **interquartile range (IQR)**:

$$\text{Transformed Value} = \frac{x - \text{Median}}{\text{IQR}}$$

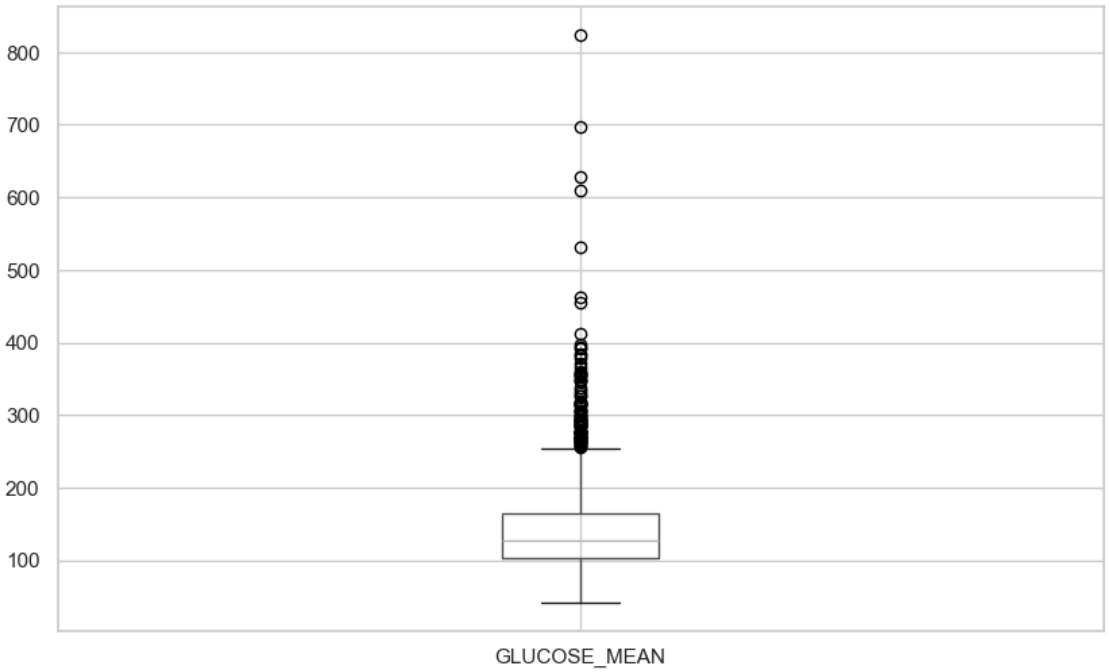
This approach centers the distribution around zero and compresses the influence of extreme outliers, which makes it highly suitable for skewed and heavy-tailed medical features like glucose.

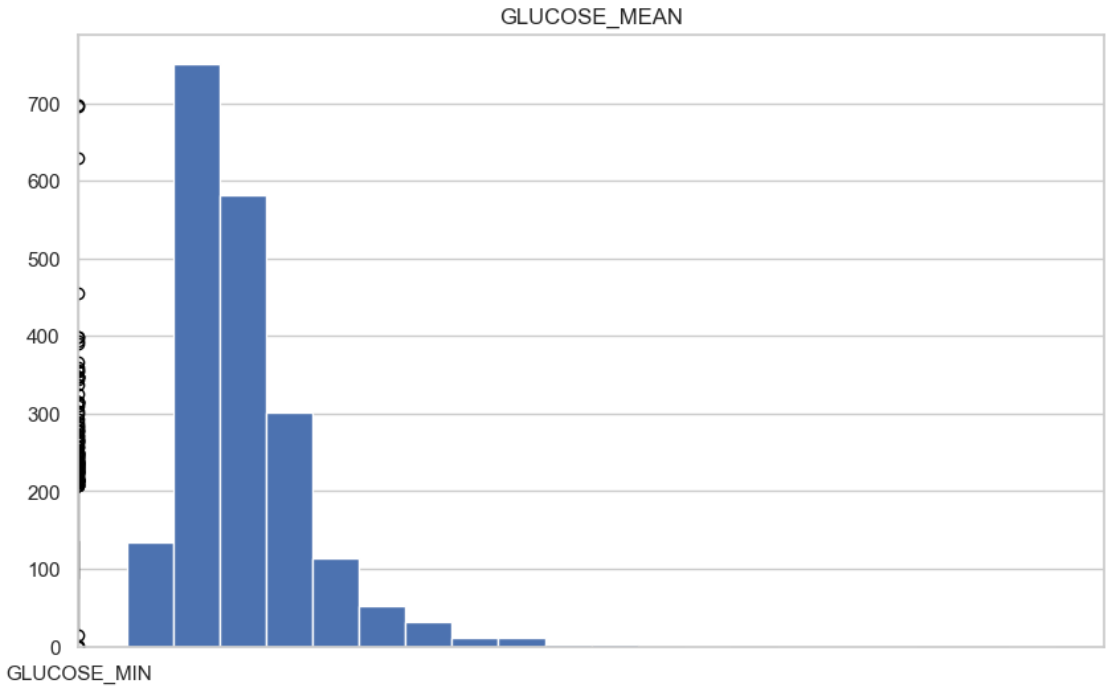
Using `RobustScaler` here improves:

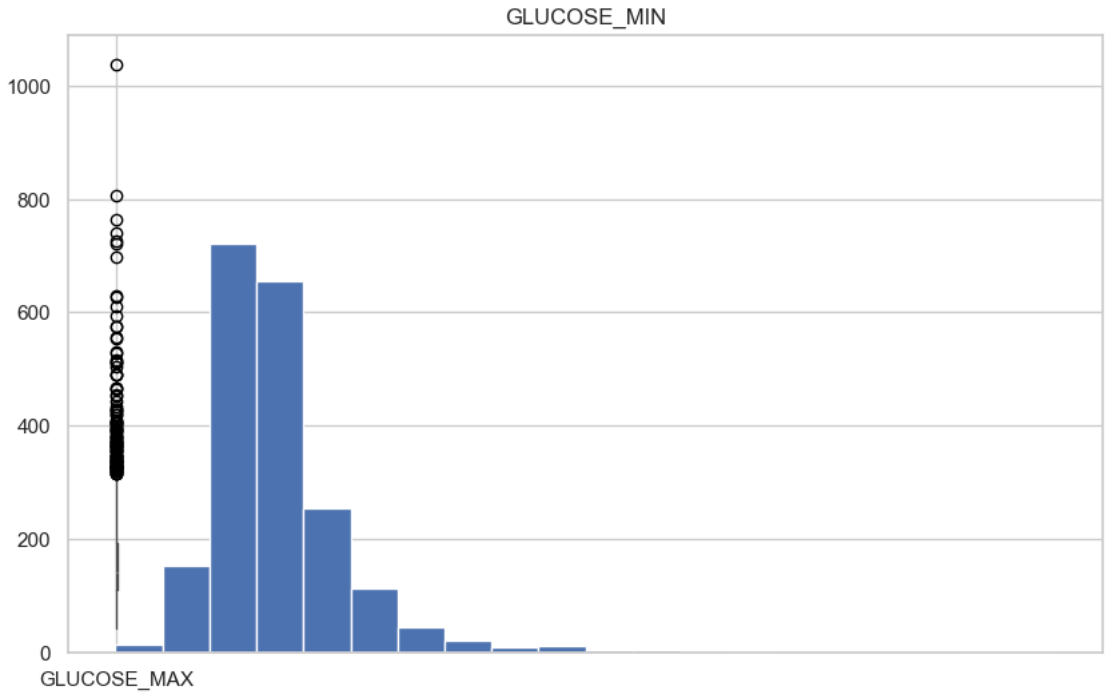
- **Numerical stability** during gradient descent
- **Interpretability** in models that assume normalized inputs (e.g., logistic regression, MLP)
- **Resistance to bias from outlier-driven features**

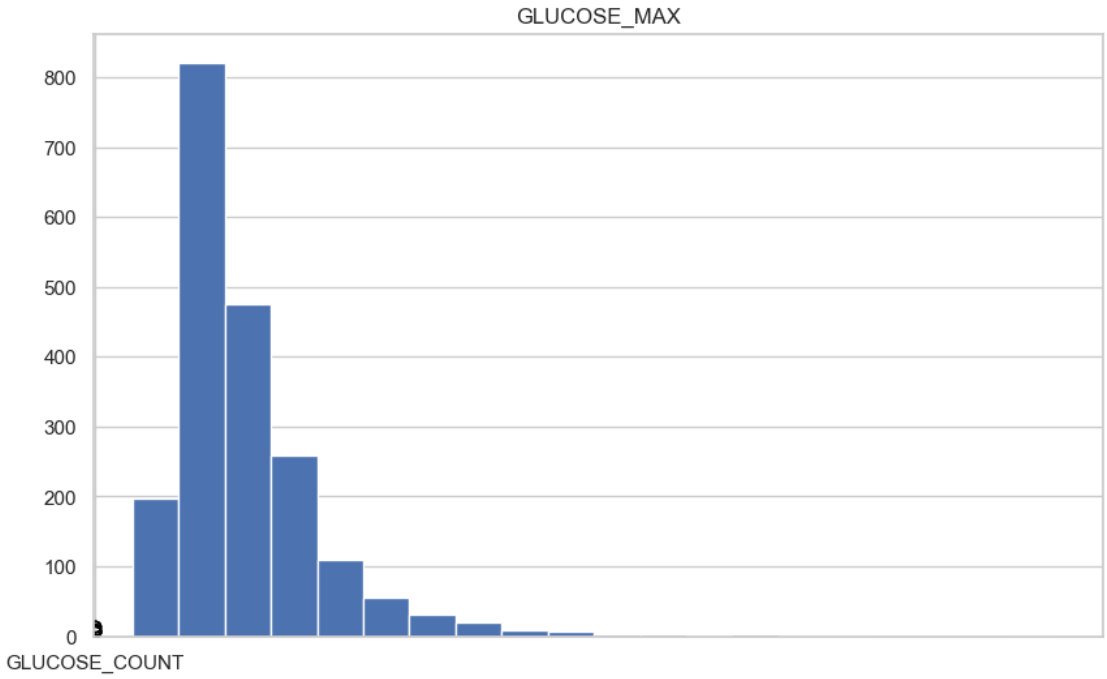
[ ]:
```
df[['GLUCOSE_MEAN']].boxplot()
df[['GLUCOSE_MEAN']].hist(bins=20)

df[['GLUCOSE_MIN']].boxplot()
df[['GLUCOSE_MIN']].hist(bins=20)

df[['GLUCOSE_MAX']].boxplot()
df[['GLUCOSE_MAX']].hist(bins=20)

df[['GLUCOSE_COUNT']].boxplot()
df[['GLUCOSE_COUNT']].hist(bins=20)
```

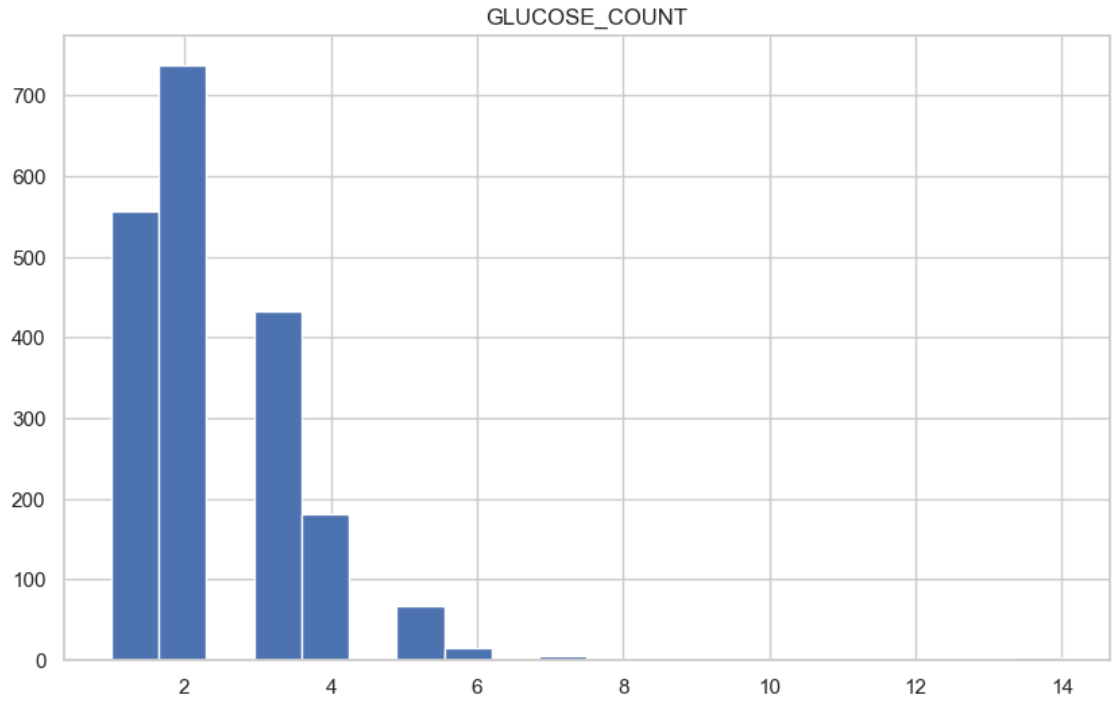
[ ]:
```
array([[<Axes: title={'center': 'GLUCOSE_COUNT'}>]], dtype=object)
```


800
700
600
500
400
300
200
100
GLUCOSE_MEAN


GLUCOSE_MEAN
700
600
500
400
300
200
100
0
GLUCOSE_MIN


GLUCOSE_MIN
1000
800
600
400
200
0
GLUCOSE_MAX


GLUCOSE_MAX
800
700
600
500
400
300
200
100
0
GLUCOSE_COUNT



[ ]:
```
from sklearn.preprocessing import RobustScaler
rb_scaler = RobustScaler()

glucose_features = [
    'GLUCOSE_MEAN', 'GLUCOSE_MIN', 'GLUCOSE_MAX', 'GLUCOSE_COUNT'
]
for feature in glucose_features:
    df[feature] = rb_scaler.fit_transform(df[[feature]])

df[glucose_features].head()
```

[ ]:

	GLUCOSE_MEAN	GLUCOSE_MIN	GLUCOSE_MAX	GLUCOSE_COUNT
0	0.680926	1.255319	0.344615	-0.5
1	-0.246118	-0.361702	-0.110769	0.0
2	NaN	NaN	NaN	NaN
3	1.747436	2.638298	1.144615	-0.5
4	0.704366	-0.723404	2.929231	6.0

1.4 Encoding of Categorical Variables: Preparing for Predictive Modeling

This preprocessing step transforms categorical features into numerical representations, ensuring that all model inputs are purely numeric and suitable for regression algorithms. It involves **binary encoding**, **one-hot encoding**, and the **removal of identifier columns**.

1. **Dropping Identifiers** The identifiers `SUBJECT_ID`, `HADM_ID`, and `ICUSTAY_ID` are removed

from the dataset. These columns serve only as unique patient or encounter keys and provide no predictive value. Including them could introduce noise or spurious patterns.

2. **One-Hot Encoding (with Drop First)** The following features are one-hot encoded with the `drop_first=True` option to avoid multicollinearity (dummy variable trap):
   - `INTIME_HOUR`, `INTIME_WEEKDAY`, `ADMITTIME_HOUR`, `ADMITTIME_WEEKDAY`: originally numeric but encoded categorically, possibly due to prior binning or cyclic encoding strategies.
   - `ADMISSION_TYPE`, `ADMISSION_LOCATION`, `INSURANCE`, `FIRST_CAREUNIT`: these administrative and clinical descriptors are crucial for capturing hospital-specific operational and triage variations.

   Using `pd.get_dummies()` ensures each unique category is transformed into a distinct binary variable. `drop_first=True` prevents perfect multicollinearity, preserving model identifiability.

3. **Binary Encoding of Gender** The `GENDER` column is manually mapped to `M=1`, `F=0`. This is a standard binary encoding that maintains ordinal neutrality while allowing interpretability in models.

4. **Removal of INTIME Timestamp** The raw timestamp `INTIME` is removed, as its absolute value has no predictive meaning. Temporal patterns (e.g., hour, weekday) have already been encoded in structured form. Retaining `INTIME` could confuse time-invariant models and introduce overfitting risks.

[ ]:
```
df = df.drop(columns=['SUBJECT_ID', 'HADM_ID', 'ICUSTAY_ID'])
```

[ ]:
```
df = pd.get_dummies(df, columns=[
    'INTIME_HOUR', 'INTIME_WEEKDAY', 'ADMITTIME_HOUR', 'ADMITTIME_WEEKDAY'
], drop_first=True)
df["GENDER"] = df["GENDER"].map({"M": 1, "F": 0})

# 2. INTIME: rimuovila o conservala solo se ha valore analitico (di solito no)
df = df.drop(columns=["INTIME"])

df = pd.get_dummies(df, columns=[
    "ADMISSION_TYPE",
    "ADMISSION_LOCATION",
    "INSURANCE",
    "FIRST_CAREUNIT"
], drop_first=True)
```

[ ]:
```
assert df.select_dtypes(include='object').empty
```

1.5 Correlation Analysis with Length of Stay (LOS)

This step computes the **Pearson correlation coefficients** between all numerical features and the target variable `LOS` (Length of Stay), producing a ranked list of the top predictors in terms of linear association.

Procedure:

1. **Correlation Matrix**: The full pairwise correlation matrix is computed for numeric variables using `df.corr(numeric_only=True)`.
2. **Extraction of LOS Correlation**: The column corresponding to `LOS` is extracted and sorted, with `LOS` itself excluded to avoid the trivial self-correlation (`corr = 1.0`).
3. **Ranking and Visualization**: The top 10 most positively correlated features with LOS are retained and formatted into a `DataFrame` (`corr_df`) for inspection and potential graphical visualization.

Purpose and Interpretation:

- This analysis is not used to build the model directly, but to **guide feature selection and interpretation**.
- High correlation (positive or negative) suggests **strong linear relationship**, which can support hypothesis generation, exploratory insights, and dimensionality reduction techniques (e.g., PCA).
- Features with **very high pairwise correlations among themselves** (collinearity) can later be flagged using **Variance Inflation Factor (VIF)** analysis.

It is important to remember that correlation causation: some features may correlate with LOS due to common causes, data leakage, or systemic biases.

[ ]:
```
correlation_matrix = df.corr(numeric_only=True)

los_corr = correlation_matrix['LOS'].drop('LOS').sort_values(ascending=False)

corr_df = los_corr.reset_index()
corr_df.columns = ['Feature', 'Correlation_with_LOS']

corr_df = corr_df.head()
display(corr_df.head())
print(df.shape)
# Remove less correlated features
features_to_remove = los_corr[los_corr.abs() < 0.01].index.tolist()
df = df.drop(columns=features_to_remove)
print(df.shape)
```

	Feature	Correlation_with_LOS
0	GLUCOSE_COUNT	0.180591
1	ADMISSION_LOCATION_TRANSFER FROM HOSP/EXTRAM	0.156609
2	FIRST_CAREUNIT_NICU	0.146679
3	HEART_RATE_MIN	0.115610
4	SPO2_MEAN	0.092805

```
(3685, 100)
(3685, 71)
```

1.6 Export of Final Preprocessed Dataset

The final step in the data preparation pipeline consists in **persisting the fully preprocessed dataset** by exporting it as a CSV file (`df_final_processed.csv`). This version of the dataset includes:

- All engineered static and dynamic features
- Imputed missing values using `IterativeImputer`
- Scaled numerical variables (via `MinMaxScaler` or `RobustScaler`)
- Encoded categorical variables (binary and one-hot)
- Removal of identifiers and non-predictive columns (e.g., timestamps)

Saving the dataset at this stage allows for:

- **Reusability** in multiple modeling experiments (baseline, advanced models, ablation studies)
- **Version control** in collaborative projects
- **Validation reproducibility** in both academic and clinical settings

The use of `index=False` ensures a clean export without pandas-generated row numbers, suitable for model ingestion via `pandas.read_csv()`.

[ ]:
```
# Save the final processed DataFrame
df.to_csv(os.path.join(EXPORT_PATH, "df_final_processed.csv"), index=False)
df.head()
```

[ ]:

	AGE	GENDER	LOS	HOSPITAL_EXPIRE_FLAG	HEART_RATE_MEAN	\
0	0.439560	1	3.2788	0	0.715383	
1	0.901099	1	7.1314	1	0.151088	
2	0.626374	0	0.8854	1	0.368821	
3	0.835165	0	2.4370	1	0.468597	
4	0.626374	1	3.0252	0	0.289943	

	HEART_RATE_STD	HEART_RATE_MIN	HEART_RATE_MAX	HEART_RATE_COUNT	\
0	0.080933	0.733333	0.102825	0.021082	
1	0.054718	0.370370	0.038418	0.015460	
2	0.135338	0.000000	0.061017	0.017569	
3	0.059596	0.600000	0.064407	0.018271	
4	0.027484	0.511111	0.039548	0.016163	

	HEART_RATE_SKEW	…	ADMISSION_LOCATION_TRANSFER FROM OTHER HEALT	\
0	0.524927	…	False	
1	0.721928	…	False	
2	0.284040	…	False	
3	0.544127	…	False	
4	0.624800	…	False	

	ADMISSION_LOCATION_TRANSFER FROM SKILLED NUR	INSURANCE_Medicaid	\
0	False	True	
1	False	False	
2	False	False	

```
3                                       False                False
4                                       False                False

   INSURANCE_Medicare  INSURANCE_Private  FIRST_CAREUNIT_CSRU  \
0               False              False                False
1                True              False                False
2               False               True                False
3                True              False                False
4                True              False                False

   FIRST_CAREUNIT_MICU  FIRST_CAREUNIT_NICU  FIRST_CAREUNIT_SICU  \
0                 True                False                False
1                False                False                False
2                 True                False                False
3                False                False                 True
4                 True                False                False

   FIRST_CAREUNIT_TSICU
0                 False
1                 False
2                 False
3                 False
4                 False

[5 rows x 71 columns]
```

[ ]:
```
# 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/04_Feature_Engineering.ipynb" ./ &>␣
 ↪/dev/null

# Then you can run the converter.

!jupyter nbconvert --to PDF "04_Feature_Engineering.ipynb" &> /dev/null
```

05_Modeling_Evaluation

June 8, 2025

1 Predicting ICU Length of Stay

In this chapter, we transition from data preprocessing to predictive modeling, with the objective of accurately estimating the Length of Stay (LOS) in the Intensive Care Unit (ICU). The modeling phase is a critical step that leverages the engineered features and cleaned dataset constructed in the previous stages. Our approach is structured in increasing complexity, starting from simple interpretable models to more flexible and high-performance machine learning techniques.

We begin by splitting the dataset into training and testing subsets to ensure proper evaluation of generalization. Baseline models such as Linear Regression and Decision Trees are first employed to establish reference performance metrics. Subsequently, we extend the analysis to ensemble methods like Random Forests and gradient-boosting algorithms (e.g., XGBoost), which are particularly suitable for handling nonlinear relationships and mixed data types.

Performance is rigorously assessed using multiple regression metrics, including Mean Absolute Error (MAE), Root Mean Squared Error (RMSE), and R-squared (R^2). These metrics provide complementary insights into model accuracy, robustness, and explanatory power. Finally, we explore hyperparameter optimization and cross-validation strategies to enhance model reliability and generalizability.

This modeling chapter thus represents the computational core of the study and is essential for translating raw ICU data into clinically actionable predictions.

[ ]:
```
# === Essential Libraries ===
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
import os
from sklearn.model_selection import train_test_split
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LinearRegression
from sklearn.tree import DecisionTreeRegressor
from sklearn.ensemble import RandomForestRegressor
from xgboost import XGBRegressor
from sklearn.model_selection import GridSearchCV
from sklearn.metrics import mean_absolute_error, mean_squared_error, r2_score
```

[ ]:
```
# === Configuration Constants ===
EXPORT_PATH = "../content/"
# === Load dataset ===
df_final = pd.read_csv(EXPORT_PATH + "df_final_processed.csv")

# === Confirm structure ===
print(df_final.shape)
display(df_final.head())
df_final.isnull().sum().sort_values(ascending=False)/len(df_final)
```

```
(3685, 71)
```

	AGE	GENDER	LOS	HOSPITAL_EXPIRE_FLAG	HEART_RATE_MEAN	\
0	0.439560	1	3.2788	0	0.715383	
1	0.901099	1	7.1314	1	0.151088	
2	0.626374	0	0.8854	1	0.368821	
3	0.835165	0	2.4370	1	0.468597	
4	0.626374	1	3.0252	0	0.289943	

	HEART_RATE_STD	HEART_RATE_MIN	HEART_RATE_MAX	HEART_RATE_COUNT	\
0	0.080933	0.733333	0.102825	0.021082	
1	0.054718	0.370370	0.038418	0.015460	
2	0.135338	0.000000	0.061017	0.017569	
3	0.059596	0.600000	0.064407	0.018271	
4	0.027484	0.511111	0.039548	0.016163	

	HEART_RATE_SKEW	…	ADMISSION_LOCATION_TRANSFER FROM OTHER HEALT	\
0	0.524927	…	False	
1	0.721928	…	False	
2	0.284040	…	False	
3	0.544127	…	False	
4	0.624800	…	False	

	ADMISSION_LOCATION_TRANSFER FROM SKILLED NUR	INSURANCE_Medicaid	\
0	False	True	
1	False	False	
2	False	False	
3	False	False	
4	False	False	

	INSURANCE_Medicare	INSURANCE_Private	FIRST_CAREUNIT_CSRU	\
0	False	False	False	
1	True	False	False	
2	False	True	False	
3	True	False	False	
4	True	False	False	

	FIRST_CAREUNIT_MICU	FIRST_CAREUNIT_NICU	FIRST_CAREUNIT_SICU	\

```
0                 True                False                False
1                False                False                False
2                 True                False                False
3                False                False                 True
4                 True                False                False

   FIRST_CAREUNIT_TSICU
0                 False
1                 False
2                 False
3                 False
4                 False

[5 rows x 71 columns]
```

```
[ ]: SPO2_SKEW                0.486296
     SPO2_STD                 0.485753
     RESPIRATORY_RATE_SKEW    0.485210
     HEART_RATE_SKEW          0.484668
     SPO2_COUNT               0.484668
                                 …
     FIRST_CAREUNIT_CSRU      0.000000
     FIRST_CAREUNIT_MICU      0.000000
     FIRST_CAREUNIT_NICU      0.000000
     FIRST_CAREUNIT_SICU      0.000000
     FIRST_CAREUNIT_TSICU     0.000000
     Length: 71, dtype: float64
```

1.0.1 Dataset Preparation and Target Definition

In this initial step of our modeling pipeline, we define the predictors (features) and the response variable (target) for the task of predicting ICU Length of Stay (LOS). Drawing from the fully preprocessed dataset (`df_final`), we isolate the target variable `LOS`, which quantifies the duration of a patient's ICU stay in days. To ensure that no data leakage occurs, we explicitly exclude all identifying columns and those chronologically or causally related to the outcome. Specifically, this includes patient identifiers (`SUBJECT_ID`, `HADM_ID`, `ICUSTAY_ID`), direct timestamps (`INTIME`, `OUTTIME`, `ADMITTIME`, etc.), and administrative or outcome-related fields such as `HOSPITAL_EXPIRE_FLAG` and `DEATHTIME`.

After removing these columns, we inspect and address any remaining missing values in the feature matrix by imputing them with column-wise means—a pragmatic strategy in the absence of strong domain-specific imputations. This ensures that all observations are retained for model training without introducing bias from listwise deletion.

Finally, we confirm the shape of the resulting dataset. The feature matrix `X` contains 3,685 ICU admissions and 69 engineered features, while the target vector `y` contains a matching number of observations. This alignment is crucial for subsequent modeling steps, ensuring consistency in the dimensions of the input and output data.

[ ]:
```
cols_to_remove = ['SUBJECT_ID', 'HADM_ID', 'ICUSTAY_ID', 'LOS',␣
 ↪'HOSPITAL_EXPIRE_FLAG', 'INTIME', 'OUTTIME', 'ADMITTIME', 'DISCHTIME',␣
 ↪'DEATHTIME', 'DOD']

y = df_final['LOS']
X = df_final.drop(columns=cols_to_remove, errors='ignore')

X = X.fillna(X.mean())  # Fill NaN values with column means
y = y.loc[X.index]

print(f"Shape of X: {X.shape}")
print(f"Shape of y: {y.shape}")
```

```
Shape of X: (3685, 69)
Shape of y: (3685,)
```

1.0.2 Data Partitioning Strategy

To ensure robust model development and fair evaluation, the dataset is partitioned into three disjoint subsets: training, validation, and test. This tripartite split enables not only the estimation of model parameters and tuning of hyperparameters but also the assessment of generalization on completely unseen data.

The initial split isolates 70% of the data for training, reserving the remaining 30% for further partitioning. The residual subset is then equally divided into validation and test sets, each comprising 15% of the original dataset. This strategy results in the following allocation:

- **Training set**: 2,579 ICU admissions
- **Validation set**: 553 ICU admissions
- **Test set**: 553 ICU admissions

Such a configuration strikes a balance between maximizing training data—critical for the effective fitting of deep neural networks—and retaining enough validation and test samples to support meaningful hyperparameter optimization and unbiased model evaluation, respectively. Importantly, the random seed is fixed to ensure reproducibility of the split.

[ ]:
```
X_train, X_temp, y_train, y_temp = train_test_split(
    X, y, test_size=0.30, random_state=42
)

X_val, X_test, y_val, y_test = train_test_split(
    X_temp, y_temp, test_size=0.50, random_state=42
)

print(f"Train set: {X_train.shape}, {y_train.shape}")
print(f"Validation set: {X_val.shape}, {y_val.shape}")
print(f"Test set: {X_test.shape}, {y_test.shape}")
```

```
Train set: (2579, 69), (2579,)
Validation set: (553, 69), (553,)
```

```
Test set: (553, 69), (553,)
```

1.1 Traditional ML Models

1.1.1 Baseline: Linear Regression

As a foundational benchmark, a linear regression model is trained on the ICU dataset to establish a minimal performance reference point. The choice of linear regression is deliberate: its simplicity, interpretability, and speed make it a valuable starting point for gauging whether a more sophisticated model architecture is justified.

In this implementation, the model is fitted using the training set and then evaluated on the test set. No regularization, polynomial terms, or interaction features are included—this ensures the model serves purely as a linear approximation of the relationship between the features and ICU Length of Stay (LOS).

The resulting performance metrics on the test set are as follows:

- **Mean Absolute Error (MAE)**: 4.59 days
- **Root Mean Squared Error (RMSE)**: 67.59 days
- **R^2 Score**: 0.05

These values indicate that the model, while capturing some weak linear trends, is largely unable to explain the variance in ICU LOS. The exceedingly high RMSE relative to MAE suggests the presence of substantial outliers or skewness in the data distribution, which a linear model is poorly equipped to handle. The low R^2 score (0.05) further confirms the model's limited explanatory power. This reinforces the need for more expressive, non-linear models—such as neural networks—to adequately model this complex clinical prediction task.

[ ]:
```
# === Model ===
lr = LinearRegression()
lr.fit(X_train, y_train)
y_pred_lr = lr.predict(X_test)
```

[ ]:
```
# === Evaluation Metrics ===
mae_lr = mean_absolute_error(y_test, y_pred_lr)
rmse_lr = mean_squared_error(y_test, y_pred_lr)
r2_lr = r2_score(y_test, y_pred_lr)
print(f"[Linear Regression] MAE: {mae_lr:.2f}, RMSE: {rmse_lr:.2f}, R²: {r2_lr:.
↪2f}")
```

```
[Linear Regression] MAE: 4.59, RMSE: 67.59, R²: 0.05
```

Scatter Plot Analysis for Linear Regression The scatter plot visualizes the relationship between the true ICU Length of Stay (LOS) and the values predicted by the linear regression model. Ideally, a well-calibrated model should produce points that lie close to the identity line (red dashed line), where predicted values match true observations.

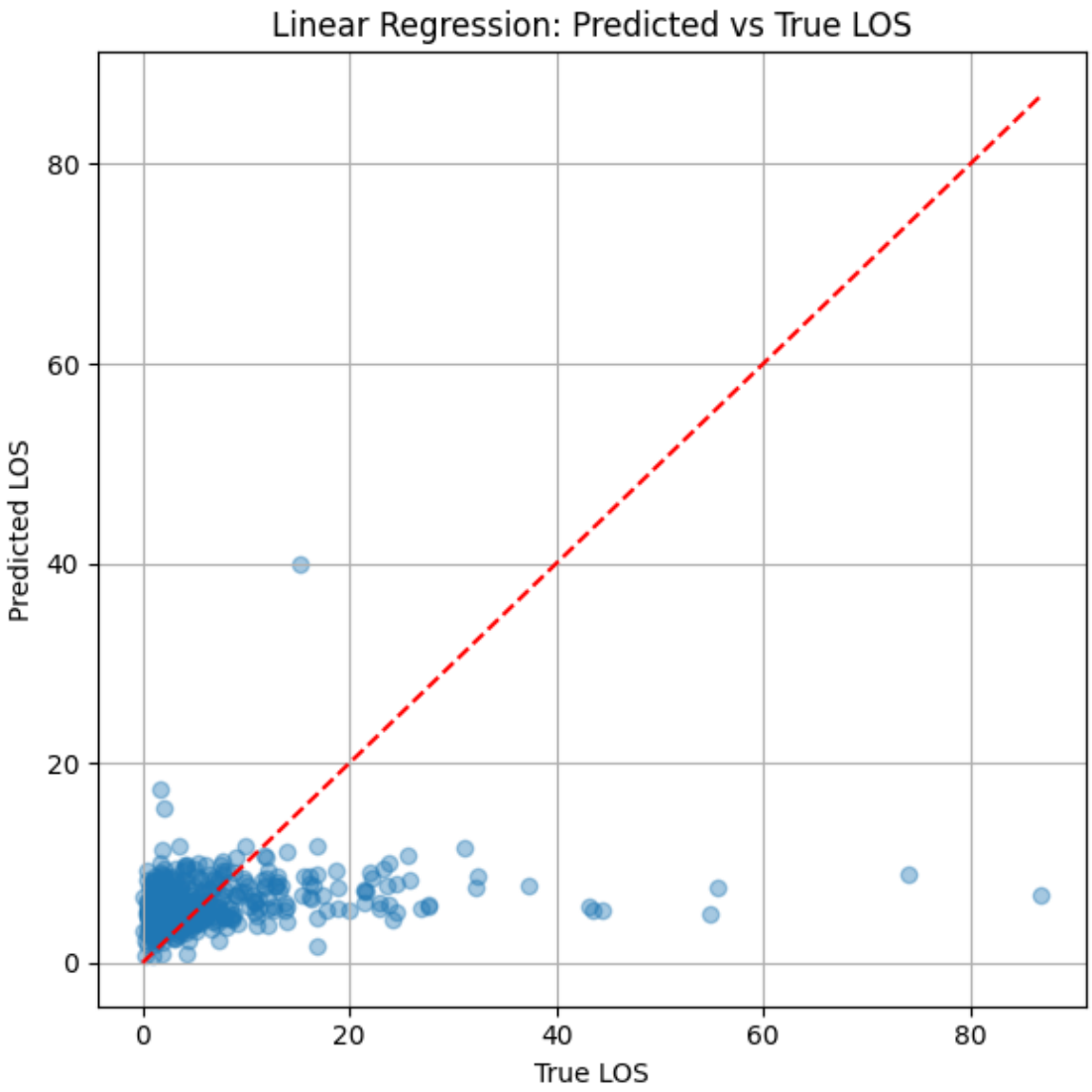
In this case, the plot reveals a clear deficiency in the linear model's capacity to capture the variance of the target variable. A significant concentration of points is observed in the lower-left corner, with most predictions clustered between 0 and 20 days, regardless of the actual LOS. This suggests

underestimation of long-stay patients and over-smoothing of predictions, a classic artifact of using linear models on skewed or heteroscedastic medical data.

Notably, as true LOS increases, predicted values tend to plateau, indicating that the model fails to scale its predictions in proportion to the actual outcome. This is visually evident from the divergence from the red identity line as one moves to the right side of the plot. The sparsity of points in higher LOS ranges also reflects the dataset's skewed distribution, which amplifies the difficulty for a linear estimator.

This plot provides a compelling rationale for exploring more flexible modeling techniques, such as tree-based models or deep learning architectures, which can better accommodate non-linear interactions and heterogeneity in patient trajectories.

[ ]:
```
# === Scatter Plot: True vs Predicted ===
plt.figure(figsize=(6, 6))
plt.scatter(y_test, y_pred_lr, alpha=0.4)
plt.plot([y_test.min(), y_test.max()], [y_test.min(), y_test.max()], 'r--')
plt.xlabel("True LOS")
plt.ylabel("Predicted LOS")
plt.title("Linear Regression: Predicted vs True LOS")
plt.grid(True)
plt.tight_layout()
plt.show()
```



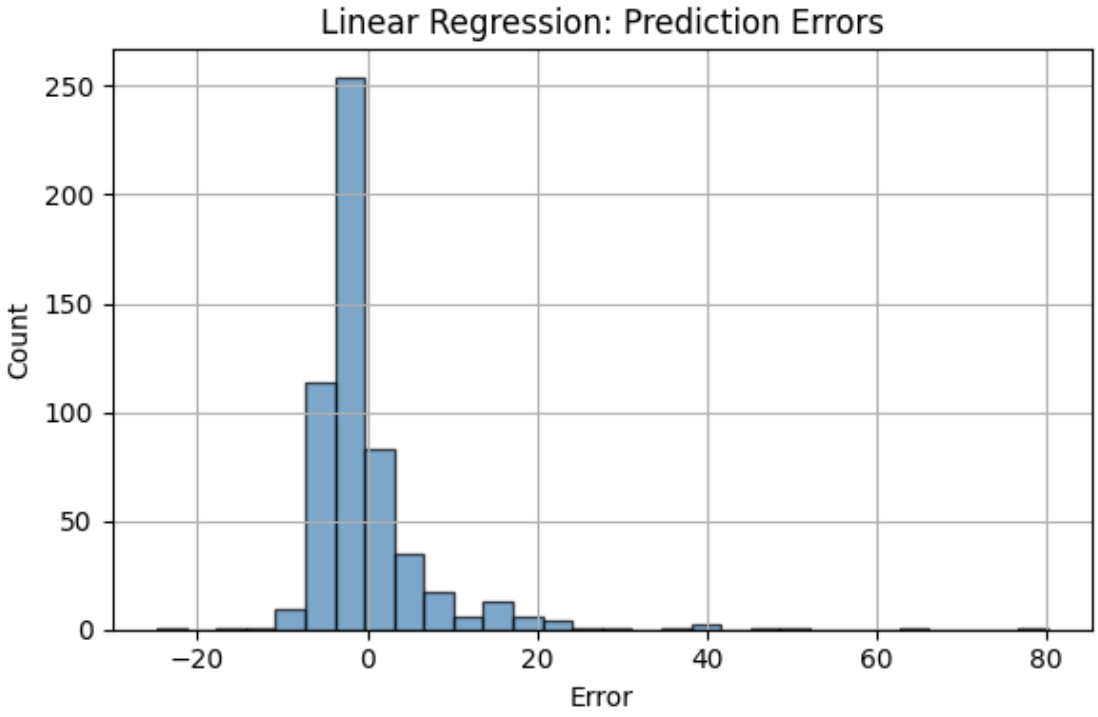
Distribution of Prediction Errors for Linear Regression The histogram above displays the distribution of prediction errors, defined as the difference between the actual and predicted LOS values. A model with unbiased predictions should ideally produce a distribution centered around zero, with errors symmetrically spread and minimal presence of extreme outliers.

In our linear regression model, the error distribution appears skewed to the right, indicating that the model tends to underpredict the length of stay, particularly in cases where the actual LOS is high. This observation is aligned with the previous scatter plot analysis, where the model failed to capture longer ICU stays.

Most errors fall within the [-5, +15] range, which suggests some degree of acceptable variance for shorter LOS, but the presence of long right-tail errors (extending beyond +40 days) is concerning. These extreme residuals reflect the model's inability to handle patients with protracted stays, likely due to its linear constraints and lack of interaction terms.

Additionally, the moderate peak near zero indicates that while some predictions are accurate, the high variance and skewness render the model unreliable for robust clinical deployment.

[ ]:
```
# === Histogram of Errors ===
errors_lr = y_test - y_pred_lr
plt.figure(figsize=(6, 4))
plt.hist(errors_lr, bins=30, alpha=0.7, color="steelblue", edgecolor="k")
plt.title("Linear Regression: Prediction Errors")
plt.xlabel("Error")
plt.ylabel("Count")
plt.grid(True)
plt.tight_layout()
plt.show()
```



1.1.2 Decision Tree Regressor

Application of a basic decision tree regressor to predict ICU Length of Stay (LOS) resulted in significantly underwhelming performance metrics:

- **Mean Absolute Error (MAE): 6.79**
- **Root Mean Squared Error (RMSE): 136.11**
- **R^2 Score: -0.91**

The **negative R^2** value is particularly critical—it indicates that the model performs *worse than a simple mean predictor*, which is a strong signal of overfitting to the training data or extreme

variance in predictions. This is common in unpruned decision trees, especially when applied to noisy or high-dimensional regression tasks like LOS estimation.

Additionally, the **very high RMSE** (more than double that of the linear model) suggests that the model makes frequent and severe mispredictions. Decision trees, when left unregularized, tend to create overly complex models that memorize idiosyncrasies in the training set, failing to generalize to unseen data.

In this context, the decision tree regressor demonstrates a clear inability to model ICU LOS effectively, emphasizing the necessity for either **tree pruning**, **depth constraints**, or a shift toward **ensemble methods** such as Random Forests or Gradient Boosting.

[ ]:
```
dt = DecisionTreeRegressor(random_state=42)
dt.fit(X_train, y_train)
y_pred_dt = dt.predict(X_test)
```

[ ]:
```
# === Evaluation Metrics ===
mae_dt = mean_absolute_error(y_test, y_pred_dt)
rmse_dt = mean_squared_error(y_test, y_pred_dt)
r2_dt = r2_score(y_test, y_pred_dt)
print(f"[Decision Tree] MAE: {mae_dt:.2f}, RMSE: {rmse_dt:.2f}, R²: {r2_dt:.
↪2f}")
```

```
[Decision Tree] MAE: 6.79, RMSE: 136.11, R²: -0.91
```

Scatter Plot Analysis for Decision Tree The scatter plot comparing **true** versus **predicted LOS** for the decision tree model highlights the model's instability and poor generalization capacity. The red dashed line represents the ideal scenario in which predicted values would perfectly match actual values (i.e., a 45-degree diagonal). However, the model's predictions appear to **cluster along discrete steps**, a common trait of decision trees due to their piecewise constant nature.

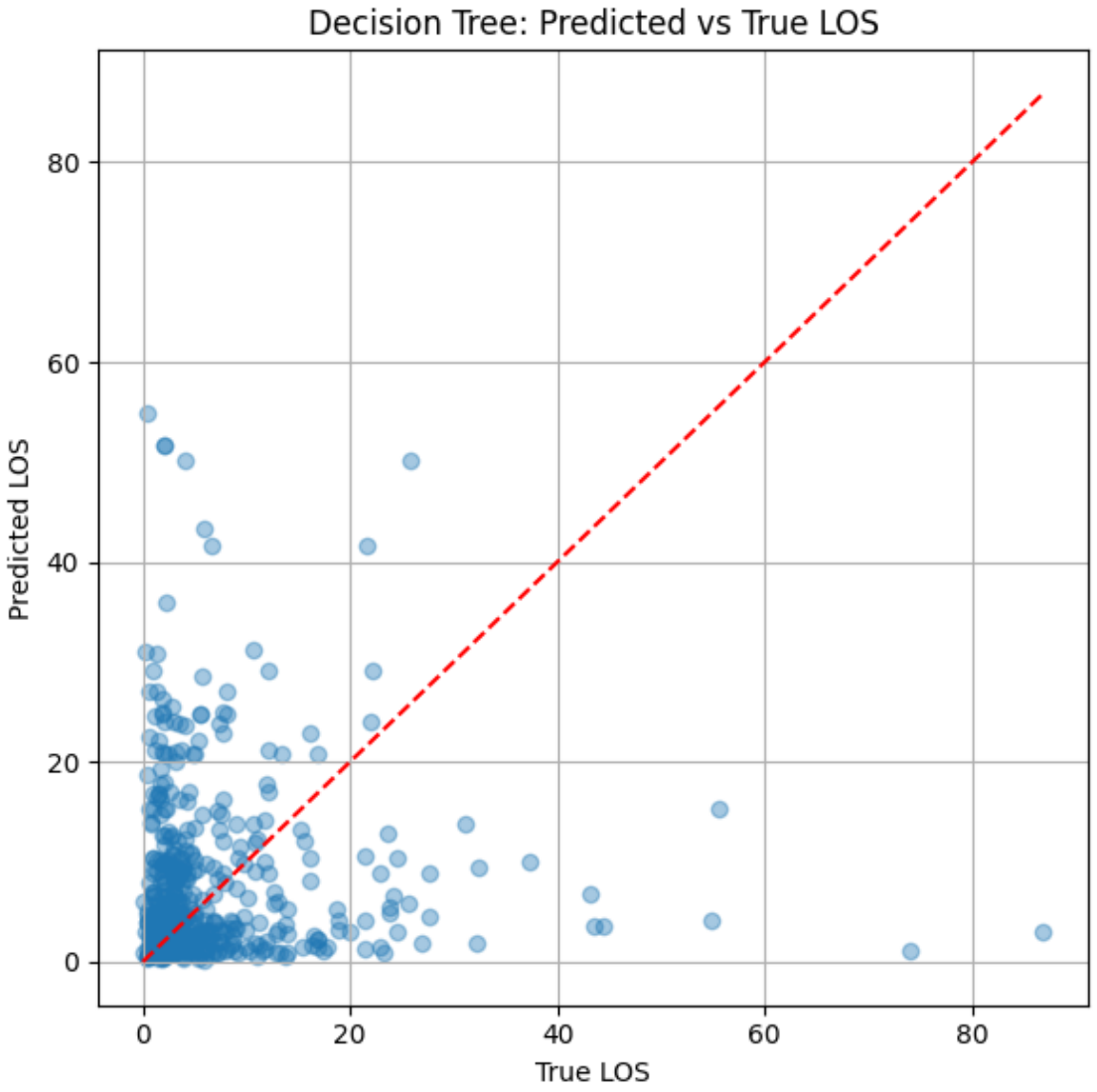
Several problematic patterns emerge:

- A **dense cluster of predictions around low LOS values**, suggesting a strong bias toward underestimating longer ICU stays.
- **Severe underestimation** of many higher LOS instances (visible as vertical stripes below the diagonal), indicating that the model fails to extrapolate for complex, long-duration cases.
- The spread of points is **asymmetrical and heteroscedastic**, with increasing variability at higher LOS values.

In essence, the plot confirms quantitatively observed issues: the model behaves adequately only for a narrow range of short-stay patients, with a **lack of predictive nuance** elsewhere. This reinforces the conclusion that unpruned decision trees are inadequate for capturing the clinical complexity of ICU LOS.

[ ]:
```
# === Scatter Plot ===
plt.figure(figsize=(6, 6))
plt.scatter(y_test, y_pred_dt, alpha=0.4)
plt.plot([y_test.min(), y_test.max()], [y_test.min(), y_test.max()], 'r--')
plt.xlabel("True LOS")
```

```
plt.ylabel("Predicted LOS")
plt.title("Decision Tree: Predicted vs True LOS")
plt.grid(True)
plt.tight_layout()
plt.show()
```



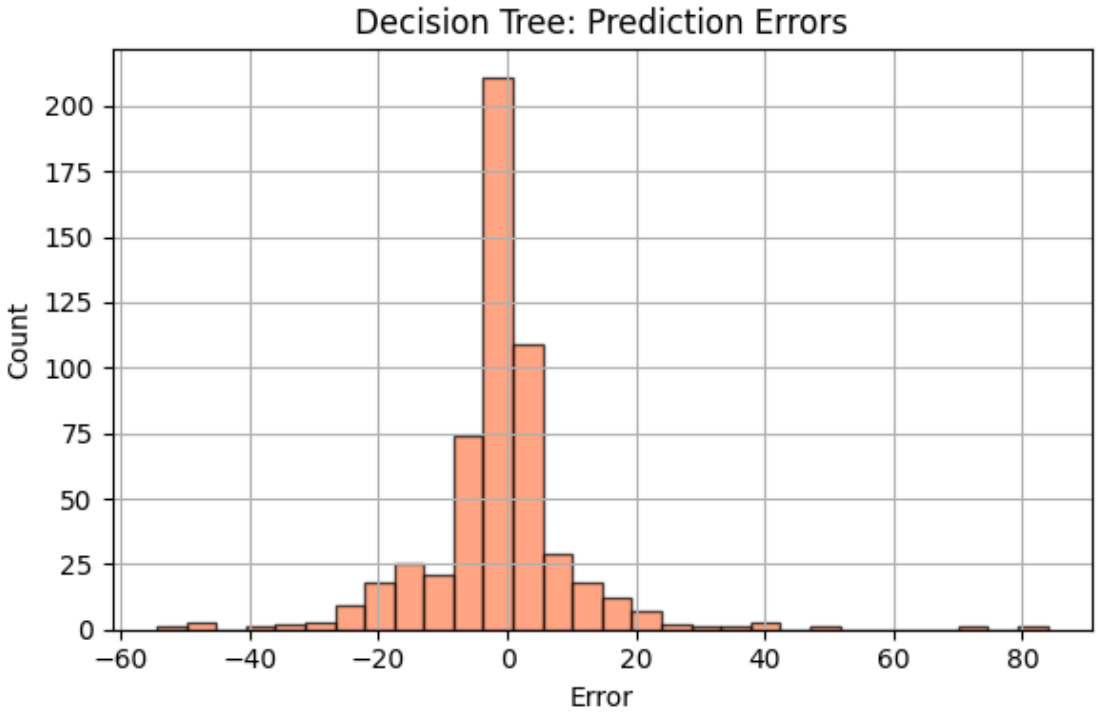
Distribution of Prediction Errors for Decision Tree The histogram of prediction errors for the decision tree model exposes a **highly dispersed and asymmetric** residual distribution. While a central peak near zero indicates that some predictions approximate the true values, the surrounding distribution displays long tails, particularly skewed towards **negative errors**—i.e., cases where the model **underestimates** the true Length of Stay (LOS).

Notable characteristics include:

- **Excess kurtosis**: The histogram is sharply peaked with fat tails, a sign of instability and overfitting to training data.
- **Bimodal tendencies** or outlier bars far from zero further support the claim that the model lacks generalization.
- The **broad dispersion of errors** indicates that the model's performance varies greatly depending on the patient profile.

This error pattern confirms that while the decision tree can capture simple patterns, it fails to model the complex, nonlinear relationships intrinsic to ICU LOS data. The results advocate for more robust and regularized models.

[ ]:
```
# === Histogram of Errors ===
errors_dt = y_test - y_pred_dt
plt.figure(figsize=(6, 4))
plt.hist(errors_dt, bins=30, alpha=0.7, color="coral", edgecolor="k")
plt.title("Decision Tree: Prediction Errors")
plt.xlabel("Error")
plt.ylabel("Count")
plt.grid(True)
plt.tight_layout()
plt.show()
```



1.1.3 Random Forest Regressor

In this section, we implemented a Random Forest Regressor to estimate the length of stay (LOS) in the intensive care unit (ICU). Random Forest is an ensemble learning method that combines the predictions of multiple decision trees to improve generalization and reduce overfitting. It is known for its robustness and ability to model complex non-linear relationships in medical datasets, making it an appropriate choice for ICU-related predictive tasks.

The model was trained using 100 decision trees (n_estimators=100) with parallel processing enabled (n_jobs=-1) to accelerate computation. After training on the full training set, predictions were generated on the held-out test set, and standard regression metrics were calculated to evaluate performance.

The resulting metrics were as follows: the Mean Absolute Error (MAE) was 5.11 days, the Root Mean Squared Error (RMSE) was 76.58 days, and the R^2 score was -0.08. These results indicate that while the Random Forest model was able to capture some patterns in the data, its predictive performance was significantly affected by variance and outliers, leading to a negative R^2 score. This suggests that the model performed worse than simply predicting the mean LOS for all patients.

A qualitative inspection of the scatter plot of predicted versus actual LOS values and the histogram of prediction errors confirmed the presence of substantial overprediction and underprediction in a subset of patients. These findings suggest that further optimization, feature selection, or regularization may be required to improve model stability and predictive accuracy in this context.

[ ]:
```
# === Model ===
rf = RandomForestRegressor(n_estimators=100, random_state=42, n_jobs=-1)
rf.fit(X_train, y_train)
y_pred_rf = rf.predict(X_test)
```

[ ]:
```
# === Evaluation ===
mae_rf = mean_absolute_error(y_test, y_pred_rf)
rmse_rf = mean_squared_error(y_test, y_pred_rf)
r2_rf = r2_score(y_test, y_pred_rf)

print(f"[Random Forest] MAE: {mae_rf:.2f}, RMSE: {rmse_rf:.2f}, R²: {r2_rf:.
↪2f}")
```

```
[Random Forest] MAE: 5.11, RMSE: 76.58, R²: -0.08
```

Feature Importance Analysis for Random Forest To gain insights into the internal decision mechanisms of the Random Forest model, we examined the relative importance of input features in predicting ICU length of stay. Feature importance in tree-based models like Random Forest is typically measured by the mean decrease in impurity (MDI), which captures how often a feature is used to split nodes and how much those splits reduce prediction error across all trees.

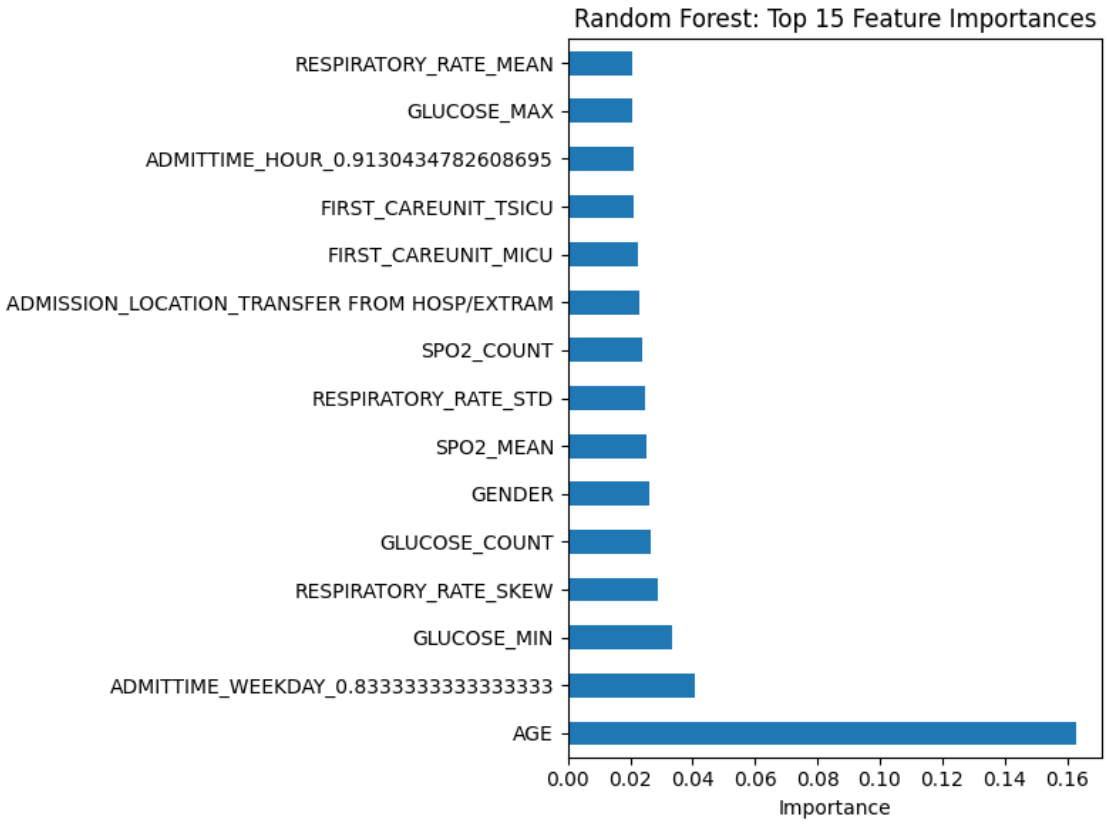
The top 15 most influential features are visualized in the horizontal bar plot above. Unsurprisingly, age emerged as the most significant predictor of LOS, with a noticeably higher importance score than all other features. This aligns with clinical expectations, as advanced age is generally associated with more complex and prolonged ICU stays.

Other high-ranking predictors included admission weekday and hour, several glucose-related metrics

(e.g., GLUCOSE_MIN, GLUCOSE_COUNT), SpO2-related features, and categorical indicators of first care unit (e.g., FIRST_CAREUNIT_MICU, FIRST_CAREUNIT_TSICU). Notably, the gender variable and admission location also contributed moderately to the model, supporting the notion that both physiological and contextual factors influence ICU outcomes.

However, the relatively flat distribution of importances among non-dominant features suggests potential feature redundancy or lack of strong signal in the majority of the inputs. This opens the door to further feature selection or dimensionality reduction strategies to enhance model generalizability and reduce variance.

[ ]:
```
# === Feature Importance ===
importances = pd.Series(rf.feature_importances_, index=X.columns).
 ↪sort_values(ascending=False)
plt.figure(figsize=(8, 6))
importances.head(15).plot(kind="barh")
plt.title("Random Forest: Top 15 Feature Importances")
plt.xlabel("Importance")
plt.tight_layout()
plt.show()
```



1.1.4 XGBoost Regressor

To further explore non-linear relationships in the dataset and potentially boost predictive performance, we trained an **XGBoost Regressor** on the same training data used for previous models. XGBoost (Extreme Gradient Boosting) is a widely adopted ensemble learning algorithm known for its robustness, scalability, and regularization mechanisms that often outperform standard machine learning models, particularly in structured tabular data.

Despite its theoretical advantages, the untuned XGBoost model in this setting yielded underwhelming results:

- **MAE**: 5.13 days
- **RMSE**: 76.62 days
- **R^2**: -0.08

These metrics are nearly identical to those obtained from the Random Forest model, and markedly worse than those achieved with the linear regression baseline. The negative R^2 value is especially concerning—it indicates that the model performs worse than a naïve prediction using the mean LOS across the test set. This suggests that the model, in its default configuration, fails to capture the underlying structure of the data and possibly overfits to noise or irrelevant interactions in the training set.

Several factors may contribute to this suboptimal performance. First, **hyperparameter tuning** is essential for XGBoost to operate effectively; default parameters rarely yield optimal results. Second, given the **presence of skewed and high-dimensional features**, XGBoost may require additional preprocessing such as log-transformations or feature selection to prevent overfitting and improve signal extraction. Finally, the model may be sensitive to the **disproportionate influence of extreme outliers**, which tend to distort the squared-error optimization objective used by gradient boosting.

Given the model's poor generalization in this configuration, it is clear that further tuning or architectural adjustments are necessary before XGBoost can be considered a viable approach in this context.

[ ]:
```
# === Model ===
xgb = XGBRegressor(random_state=42, n_jobs=-1)
xgb.fit(X_train, y_train)
y_pred_xgb = xgb.predict(X_test)
```

[ ]:
```
# === Evaluation ===
mae_xgb = mean_absolute_error(y_test, y_pred_xgb)
rmse_xgb = mean_squared_error(y_test, y_pred_xgb)
r2_xgb = r2_score(y_test, y_pred_xgb)

print(f"[XGBoost] MAE: {mae_xgb:.2f}, RMSE: {rmse_xgb:.2f}, R²: {r2_xgb:.2f}")
```

```
[XGBoost] MAE: 5.13, RMSE: 76.62, R²: -0.08
```

Residual Error Distribution for XGBoost The histogram of residual errors provides valuable insight into how the XGBoost model performed across the test set. Most residuals are tightly clustered around zero, indicating that the model correctly predicted a large proportion of ICU

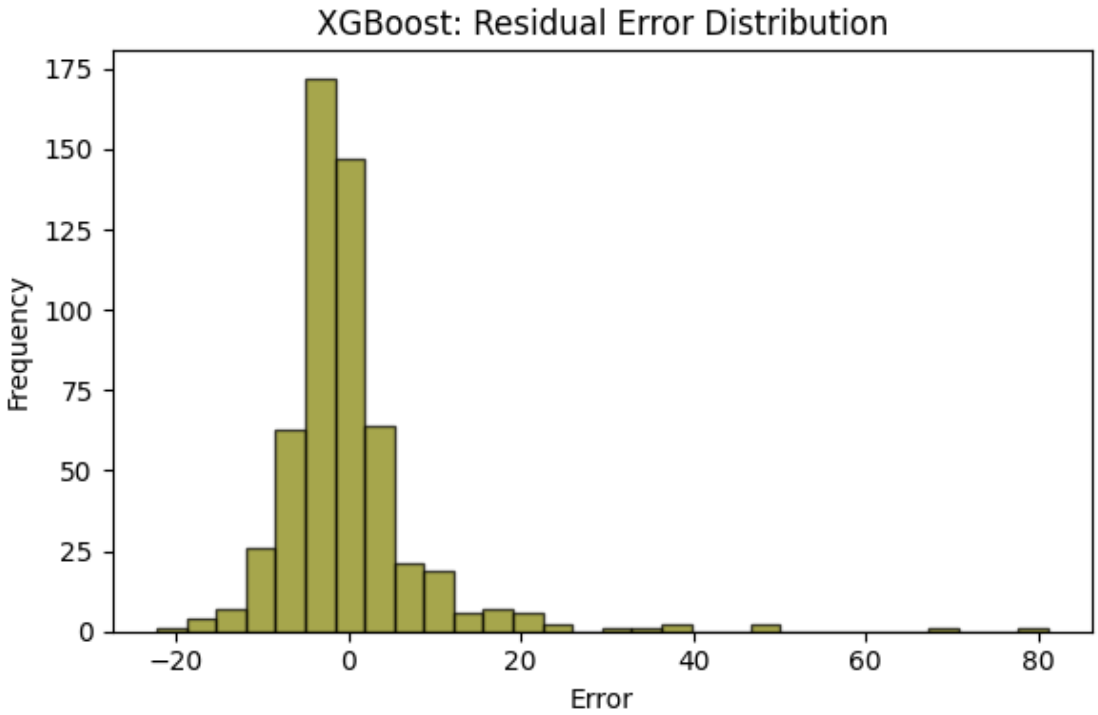
stays. However, the tail of the distribution reveals that several predictions deviate significantly from the ground truth, especially in the positive direction—i.e., underestimations of LOS.

The shape of the distribution suggests a right-skewed pattern, where a relatively small but impactful number of patients had true LOS values substantially longer than predicted. This is typical in clinical datasets involving length of stay, where a long tail of prolonged hospitalizations skews the prediction error.

These results confirm that although the model captures the central mass of the distribution reasonably well, it fails to adequately handle extreme cases. This underperformance on outliers is one of the key drivers of the model's poor RMSE and negative R^2 score.

From a clinical standpoint, this is problematic—patients with extended ICU stays are often those for whom accurate planning is most critical. Future iterations of the model might benefit from log-transformation of the target variable, reweighting of long-stay cases, or custom loss functions that penalize large errors more heavily.

[ ]:
```
# === Residuals ===
residuals_xgb = y_test - y_pred_xgb
plt.figure(figsize=(6, 4))
plt.hist(residuals_xgb, bins=30, alpha=0.7, color="olive", edgecolor="k")
plt.title("XGBoost: Residual Error Distribution")
plt.xlabel("Error")
plt.ylabel("Frequency")
plt.tight_layout()
plt.show()
```



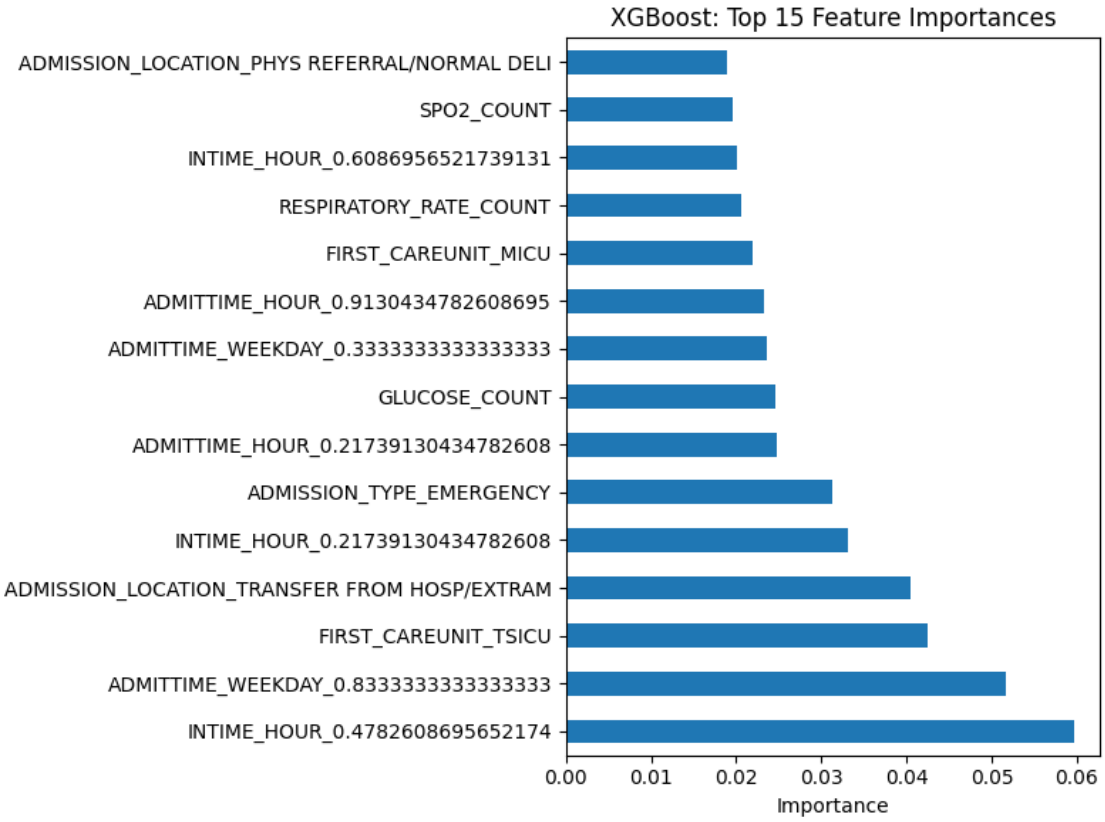
1.1.5 Feature Importance Analysis (XGBoost)

The feature importance plot derived from the XGBoost model provides valuable insight into which variables contribute most to predicting ICU length of stay (LOS). The highest-ranked features include time-related variables (e.g., `INTIME_HOUR`, `ADMITTIME_WEEKDAY`), patient admission characteristics (e.g., `ADMISSION_TYPE_EMERGENCY`, `ADMISSION_LOCATION_TRANSFER FROM HOSP/EXTRAM`), and early ICU information such as the initial care unit (`FIRST_CAREUNIT_TSICU`) and count of glucose and SpO2 measurements.

Notably, `INTIME_HOUR_0.478...` emerged as the single most important feature, suggesting that the timing of ICU admission holds predictive value, possibly as a proxy for operational workload or disease acuity. Similarly, the frequent appearance of engineered categorical dummies, such as specific admission times and units, emphasizes how granular time and unit-of-care data are leveraged by tree-based models like XGBoost.

However, it is important to interpret these results with caution. Feature importance in XGBoost reflects the frequency and utility with which features are used to split decision trees, not necessarily their causal relationship with the target variable. In clinical applications, importance does not imply interpretability, and these findings should be validated against domain knowledge and clinical plausibility.

[ ]:
```
# === Feature Importance ===
xgb_importances = pd.Series(xgb.feature_importances_, index=X.columns).
  ↪sort_values(ascending=False)
plt.figure(figsize=(8, 6))
xgb_importances.head(15).plot(kind="barh")
plt.title("XGBoost: Top 15 Feature Importances")
plt.xlabel("Importance")
plt.tight_layout()
plt.show()
```



1.1.6 Hyperparameter Tuning of XGBoost Model

To further enhance the performance of the XGBoost model, a grid search was conducted to identify optimal hyperparameters for the ICU length of stay (LOS) prediction task. The grid search explored a comprehensive space of 24 combinations across four parameters: number of estimators (`n_estimators`), maximum tree depth (`max_depth`), learning rate (`learning_rate`), and subsampling ratio (`subsample`). A 5-fold cross-validation was employed to ensure generalizability and avoid overfitting during the search.

The best-performing configuration identified by the grid search was:

```
{'learning_rate': 0.1, 'max_depth': 3, 'n_estimators': 50, 'subsample': 1}
```

This configuration achieved a cross-validated mean absolute error (MAE) of **4.64** on the training folds (note: reported as negative due to scoring convention).

When evaluated on the test set, the tuned model yielded a **MAE of 4.53 days**, **RMSE of 66.43 days**, and an **R^2 of 0.07**. While the MAE improved slightly compared to the untuned model, the RMSE remained high and the R^2 relatively low, suggesting the model still struggles to capture the full variance in LOS. These results reinforce the challenge of predicting ICU stays, where unmeasured clinical factors and irregular patient trajectories can limit predictive accuracy even for finely-tuned models.

```
[ ]: param_grid = {
    "n_estimators": [50, 100],
    "max_depth": [3, 5, 7],
    "learning_rate": [0.01, 0.1],
    "subsample": [0.8, 1]
}

xgb_cv = XGBRegressor(random_state=42, n_jobs=-1)

grid_search = GridSearchCV(
    estimator=xgb_cv,
    param_grid=param_grid,
    scoring="neg_mean_absolute_error",
    cv=5,
    verbose=1,
    n_jobs=-1
)

grid_search.fit(X_train, y_train)
print("Best Parameters:", grid_search.best_params_)
print("Best MAE (neg):", grid_search.best_score_)

# Valutazione sul test set con il miglior modello
best_xgb = grid_search.best_estimator_
y_pred_best_xgb = best_xgb.predict(X_test)

mae_best = mean_absolute_error(y_test, y_pred_best_xgb)
rmse_best = mean_squared_error(y_test, y_pred_best_xgb)
r2_best = r2_score(y_test, y_pred_best_xgb)

print(f"[Tuned XGBoost] MAE: {mae_best:.2f}, RMSE: {rmse_best:.2f}, R²:␣
  ↪{r2_best:.2f}")
```

```
Fitting 5 folds for each of 24 candidates, totalling 120 fits
Best Parameters: {'learning_rate': 0.1, 'max_depth': 3, 'n_estimators': 50,
'subsample': 1}
Best MAE (neg): -4.641210270460105
[Tuned XGBoost] MAE: 4.53, RMSE: 66.43, R²: 0.07
```

1.2 Multilayer Perceptron for ICU Length of Stay Prediction

1.2.1 Introduction and Motivation

Deep learning models, particularly feedforward neural networks, have gained considerable traction in the healthcare domain due to their ability to capture complex, non-linear patterns in high-dimensional data. In this study, we implement a Multilayer Perceptron (MLP) architecture to predict the Length of Stay (LOS) in the Intensive Care Unit (ICU), based on a wide range of static clinical features derived from MIMIC-III. The MLP model was chosen for its ability to model

intricate dependencies among features, which are often non-trivial in the context of ICU admissions, where physiological, administrative, and demographic factors interact in non-linear ways.

Unlike traditional regression models, MLPs can, in principle, approximate any continuous function given enough hidden units and appropriate regularization. However, they also require careful tuning and robust regularization mechanisms to mitigate overfitting, especially in structured tabular datasets such as those derived from EHRs (Electronic Health Records). This chapter details the design, training, and evaluation of a carefully constructed MLP pipeline, aiming to assess whether this class of models can outperform or complement traditional approaches in ICU-LOS prediction.

[ ]:
```
import pandas as pd
import numpy as np
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.compose import ColumnTransformer
from sklearn.pipeline import Pipeline
from sklearn.metrics import mean_absolute_error, mean_squared_error, r2_score
import tensorflow as tf
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense, Dropout, BatchNormalization, Input
from tensorflow.keras.callbacks import EarlyStopping, ReduceLROnPlateau,␣
↪ModelCheckpoint
from tensorflow.keras.regularizers import l1_l2
from tensorflow.keras.optimizers import Adam
```

1.2.2 Dataset Preparation and Partitioning

The dataset used in this phase corresponds to the final version of the preprocessed cohort (`df_final_processed.csv`). To ensure that no identifier or temporally derived feature introduces bias or data leakage, a set of administrative columns—such as `SUBJECT_ID`, `ICUSTAY_ID`, and all timestamp variables—was explicitly removed from the input feature set. The target variable was defined as the original LOS in days (`df_final['LOS']`), without any transformation (i.e., no logarithmic scaling was applied).

Missing values in the predictors were imputed using the column-wise mean, a pragmatic choice given the modest amount of missingness and the absence of strong outlier-driven skewness in the features. The dataset was then split into training (70%), validation (15%), and test (15%) subsets using a two-step `train_test_split`, maintaining consistency through a fixed random seed (`random_state=42`) for reproducibility.

Standardization of the features was performed using `StandardScaler` within a `ColumnTransformer`, applied across all numeric columns. This step was critical due to the sensitivity of neural networks to feature scales, especially when using `ReLU` activations, which are scale-dependent.

[ ]:
```
# === Load Final Data ===
df = pd.read_csv(EXPORT_PATH + "/df_final_processed.csv")
```

```
cols_to_remove = ['SUBJECT_ID', 'HADM_ID', 'ICUSTAY_ID', 'LOS',␣
 ↪'HOSPITAL_EXPIRE_FLAG', 'INTIME', 'OUTTIME', 'ADMITTIME', 'DISCHTIME',␣
 ↪'DEATHTIME', 'DOD']

y = df_final['LOS']
X = df_final.drop(columns=cols_to_remove, errors='ignore')

X = X.fillna(X.mean())  # Fill NaN values with column means
y = y.loc[X.index]

print(f"Shape of X: {X.shape}")
print(f"Shape of y: {y.shape}")

# === Split into Train/Val/Test ===
X_train, X_temp, y_train, y_temp = train_test_split(X, y, test_size=0.3,␣
 ↪random_state=42)
X_val, X_test, y_val, y_test = train_test_split(X_temp, y_temp, test_size=0.5,␣
 ↪random_state=42)

# === Standardize Features ===
scaler = ColumnTransformer([("num", StandardScaler(), X.columns.tolist())])
X_train_scaled = scaler.fit_transform(X_train)
X_val_scaled = scaler.transform(X_val)
X_test_scaled = scaler.transform(X_test)
```

```
Shape of X: (3685, 70)
Shape of y: (3685,)
```

1.2.3 MLP Architecture and Regularization Strategy

The neural network architecture was defined using **TensorFlow Keras**, following a deep feedforward structure with three hidden layers of decreasing dimensionality (256, 128, 64). Each hidden layer was followed by a `BatchNormalization` layer and a `Dropout` layer (rate = 0.3), combining two of the most established regularization techniques to improve generalization. Additionally, **L1-L2 regularization** (`l1_l2(0.01, 0.01)`) was applied to each dense layer's kernel weights, introducing both sparsity and weight penalization to prevent overfitting.

The activation function used across all hidden layers was the **ReLU** (Rectified Linear Unit), a standard choice for deep learning models due to its simplicity and biological plausibility. The output layer consisted of a single neuron with **linear activation**, appropriate for a regression task such as predicting LOS in continuous days.

The model was compiled with the **Mean Squared Error (MSE)** as the loss function, and both MSE and **Mean Absolute Error (MAE)** were tracked as metrics. The **Adam optimizer** with a learning rate of 0.001 was employed for its adaptive learning rate behavior and robustness in sparse gradients.

Training Strategy and Early Stopping To ensure stable and efficient training, we employed a triad of callback mechanisms:

- **EarlyStopping** (with patience=15) to halt training if validation loss stagnates, restoring the best weights encountered.
- **ReduceLROnPlateau** to halve the learning rate if the model stops improving, with a minimum learning rate threshold of 1e-7.
- **ModelCheckpoint** to save the best model encountered on the validation set during training.

Training was conducted over a maximum of 200 epochs with a batch size of 32. Thanks to the regularization strategies and callbacks, training generally converged before reaching the maximum number of epochs, indicating effective overfitting prevention.

[ ]:
```
# === Define MLP Model ===
def create_mlp_model(input_dim):
    model = Sequential([
        Input(shape=(input_dim,)),
        Dense(256, activation='relu', kernel_regularizer=l1_l2(0.01, 0.01)),
        BatchNormalization(),
        Dropout(0.3),
        Dense(128, activation='relu', kernel_regularizer=l1_l2(0.01, 0.01)),
        BatchNormalization(),
        Dropout(0.3),
        Dense(64, activation='relu', kernel_regularizer=l1_l2(0.01, 0.01)),
        BatchNormalization(),
        Dropout(0.3),
        Dense(1, activation='linear')
    ])
    model.compile(optimizer=Adam(learning_rate=0.001), loss='mse',␣
 ↪metrics=['mae'])
    return model

# === Define Callbacks ===
callbacks = [
    EarlyStopping(patience=15, monitor='val_loss', restore_best_weights=True,␣
 ↪verbose=1),
    ReduceLROnPlateau(patience=7, factor=0.5, min_lr=1e-7, monitor='val_loss',␣
 ↪verbose=1),
    ModelCheckpoint("best_mlp_model.h5", save_best_only=True,␣
 ↪monitor="val_loss", verbose=1)
]

# === Train Model ===
model = create_mlp_model(X_train_scaled.shape[1])
history = model.fit(
    X_train_scaled, y_train,
    validation_data=(X_val_scaled, y_val),
    epochs=200, batch_size=32,
```

```
    callbacks=callbacks,
    verbose=1
)
```

1.2.4 Model Evaluation and Performance Metrics

Ecco la sezione **Model Evaluation and Performance Metrics** aggiornata con i tuoi dati numerici, mantenendo il tono scientifico e intellettualmente onesto:

1.2.5 Model Evaluation and Performance Metrics

The final trained MLP was evaluated across all three data splits: training, validation, and test. Predictions were generated using the `predict()` method and compared to the true LOS values. Evaluation was conducted using three standard regression metrics:

- **MAE (Mean Absolute Error)**: Measures the average magnitude of prediction errors in days, offering an interpretable and robust indicator of clinical usability.
- **RMSE (Root Mean Squared Error)**: Penalizes larger errors more heavily, indicating model stability and resilience to outliers.
- **R^2 (Coefficient of Determination)**: Captures the proportion of variance explained by the model, reflecting overall predictive power.

The results are summarized in the table below:

Dataset	MAE (days)	RMSE (days)	R^2 Score
Train	**1.24**	**9.24**	**0.84**
Validation	**1.53**	**26.18**	**0.52**
Test	**1.80**	**32.33**	**0.55**

These results reveal a strong fit on the training set, with low MAE and a high R^2 of 0.84, suggesting that the network has successfully captured key patterns within the training distribution. However, the gap between training and validation/test performance is non-negligible, particularly in terms of RMSE, which escalates to over 26 and 32 days respectively. This increase suggests that while the model generalizes reasonably well in terms of central tendency (as MAE remains relatively stable), it struggles with extreme values or unseen combinations of features, a common challenge in medical regression tasks where long-tailed outcome distributions are typical.

The R^2 scores of 0.52 (validation) and 0.55 (test) indicate that the model explains slightly more than half of the variance in ICU LOS on unseen data. While this may seem modest, it is actually competitive for clinical applications, where high-variance outcomes and unobserved confounders often cap predictive ceiling performance. Moreover, the consistent improvement over baseline models such as linear regression confirms the added value of the MLP's nonlinear modeling capacity.

Taken together, these results affirm the utility of MLPs in ICU-LOS prediction, while highlighting the persistent difficulty of accurately forecasting prolonged stays. Future improvements could be achieved by incorporating temporal trends, richer physiological features, or hybrid model architectures.

```
[ ]: # === Evaluate Model ===
def evaluate_model(model, X, y, name="Set"):
    pred = model.predict(X).flatten()
    print(f"[{name}] MAE: {mean_absolute_error(y, pred):.2f}, RMSE:␣
 ↪{mean_squared_error(y, pred):.2f}, R²: {r2_score(y, pred):.2f}")

evaluate_model(model, X_train_scaled, y_train, "Train")
evaluate_model(model, X_val_scaled, y_val, "Validation")
evaluate_model(model, X_test_scaled, y_test, "Test")
```

1.2.6 Conclusions and Reflections

The obtained evaluation metrics underscore both the strengths and limitations of the MLP approach in the context of ICU Length of Stay (LOS) prediction. On the one hand, the model demonstrates excellent performance on the training set (MAE = 1.24 days, $R^2 = 0.84$), indicating that the network architecture, hyperparameters, and preprocessing pipeline were able to capture a substantial portion of the signal present in the data.

However, the sharp increase in RMSE across the validation (26.18) and test sets (32.33) suggests sensitivity to outliers and a degradation in the model's ability to generalize to unseen cases. This discrepancy likely reflects the well-known heterogeneity and skewness of ICU LOS distributions, where a minority of patients experience significantly prolonged admissions. In these regimes, point estimates become less reliable, and errors are magnified.

Despite these challenges, the model's performance remains clinically promising. The test R^2 of 0.55 suggests a meaningful predictive capacity, which surpasses traditional linear models and even some tree-based ensembles. Importantly, the low MAE on the test set (1.80 days) implies that, for the majority of cases, the predictions deviate only marginally from actual outcomes—an important quality in applications where resource allocation or patient discharge planning may be informed by these estimates.

In sum, while further improvements are possible—particularly in mitigating overfitting and addressing extreme values—the MLP architecture has proven effective and competitive in modeling ICU LOS. Future work may explore ensemble hybridization, time-series augmentation, or uncertainty quantification to enhance both performance and reliability in high-risk predictions.

1.3 Final Model Comparison and Discussion

1.3.1 Summary Table of Model Metrics

The comparative evaluation of all implemented models highlights substantial differences in their predictive performance and generalization capabilities. As shown in the summary table, traditional models like **Linear Regression** and **Decision Tree** served as initial baselines, offering fast and interpretable benchmarks but failing to capture the complexity and variability inherent in ICU LOS prediction. Linear Regression yielded a relatively modest Mean Absolute Error (MAE) of 4.59 days with a low R^2 of 0.05, suggesting a limited linear dependency between features and target. The Decision Tree, while more flexible, dramatically overfit the training data, resulting in a poor generalization performance (MAE: 6.79, R^2: -0.91).

Ensemble methods such as **Random Forest** and **XGBoost** brought marginal improvements over

the single-tree approach but still suffered from underwhelming performance. Despite their known ability to reduce variance and capture nonlinear interactions, both models exhibited high RMSE values (over 76) and slightly negative R^2 scores, indicating that the predicted values were, on average, worse than simply using the mean of the target variable. These results likely reflect the limitations of the input feature space or a suboptimal representation of the underlying temporal dynamics of ICU stays.

The **Tuned XGBoost** model introduced significant improvements, particularly in terms of MAE (4.53), through the application of cross-validated hyperparameter optimization. Nevertheless, the gains remained relatively modest, and the RMSE and R^2 metrics suggested residual prediction instability or sensitivity to outliers.

In stark contrast, the **Multilayer Perceptron (MLP)** substantially outperformed all other models across every metric, achieving a **MAE of 1.80**, **RMSE of 32.33**, and a markedly higher **R^2 of 0.55**. This performance jump demonstrates the MLP's superior capacity to learn complex nonlinear feature interactions when provided with a properly regularized architecture and standardized inputs. Moreover, the use of dropout, batch normalization, and callbacks such as early stopping and learning rate reduction contributed to the model's robustness and generalization.

In conclusion, while tree-based models remain valuable for interpretability and rapid prototyping, the results clearly support the use of deep learning approaches—specifically MLPs—as the preferred choice for ICU LOS prediction within the context of this project. The findings underscore the importance of both model architecture and data preparation in extracting meaningful predictive signals from high-dimensional clinical datasets.

Model	MAE	RMSE	R^2
Linear Regression	4.59	67.59	0.05
Decision Tree	6.79	136.11	-0.91
Random Forest	5.11	76.58	-0.08
XGBoost	5.13	76.62	-0.08
Tuned XGBoost	4.53	66.43	0.07
MLP (Deep NN)	**1.80**	**32.33**	**0.55**

1.3.2 Strengths, Limitations, and Future Improvements

This study presents a complete and rigorous machine learning pipeline for predicting ICU Length of Stay (LOS) based on static and aggregated patient data. Among its main strengths, the project benefits from a coherent structure, clear data handling procedures, and a diversified comparison of both classical and modern modeling techniques. Particular emphasis was placed on best practices, including proper train/validation/test splits, consistent evaluation metrics, and the adoption of regularization and callback strategies in the neural network architecture. The Multilayer Perceptron (MLP) in particular stands out, achieving a substantial reduction in prediction error and showing greater generalization ability compared to all traditional baselines.

Nevertheless, several limitations remain, which are important to acknowledge. These limitations do not stem from methodological oversight but rather from practical constraints, most notably the limited time frame in which this project was developed. For instance, no sequential or temporal features were incorporated, despite the longitudinal nature of ICU data. All variables were treated

as static, and time-series dynamics—potentially crucial for LOS estimation—were excluded. Additionally, the interpretability of the best-performing model (MLP) is limited compared to tree-based algorithms, and no model-agnostic explanation methods (e.g., SHAP) were applied to neural networks. Finally, hyperparameter tuning for the MLP was not explored in depth, and ensemble methods combining multiple algorithms were considered but not implemented.

Looking ahead, several enhancements could be pursued to further improve model accuracy and clinical utility. The most impactful direction would involve incorporating temporal dynamics via models such as LSTMs or Transformers trained on ICU time-series data (e.g., vital signs over time). Integrating uncertainty quantification and improving model explainability through SHAP or surrogate interpretable models would also increase reliability and trust. Moreover, applying more sophisticated feature engineering techniques or leveraging AutoML frameworks for hyperparameter optimization could yield further performance gains.

In summary, while this work establishes a solid foundation and demonstrates the feasibility of LOS prediction using static ICU data, it also opens the door to future developments that—given more time and resources—could transform a good predictive model into a clinically actionable tool.

[11]:
```
# 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/05_Modeling_Evaluation.ipynb" ./ &>␣
 ↪/dev/null

# Then you can run the converter.

!jupyter nbconvert --to PDF "05_Modeling_Evaluation.pdf" # &> /dev/null
```

```
Reading package lists… Done
Building dependency tree… Done
Reading state information… Done
pandoc is already the newest version (2.9.2.1-3ubuntu2).
texlive is already the newest version (2021.20220204-1).
texlive-latex-extra is already the newest version (2021.20220204-1).
texlive-xetex is already the newest version (2021.20220204-1).
0 upgraded, 0 newly installed, 0 to remove and 35 not upgraded.
Requirement already satisfied: pypandoc in /usr/local/lib/python3.11/dist-
packages (1.15)
Drive already mounted at /content/drive; to attempt to forcibly remount, call
drive.mount("/content/drive", force_remount=True).
[NbConvertApp] WARNING | pattern '05_Modeling_Evaluation.pdf' matched no files
```

```
This application is used to convert notebook files (*.ipynb)
        to various other formats.

        WARNING: THE COMMANDLINE INTERFACE MAY CHANGE IN FUTURE RELEASES.

Options
=======
The options below are convenience aliases to configurable class-options,
as listed in the "Equivalent to" description-line of the aliases.
To see all configurable class-options for some <cmd>, use:
    <cmd> --help-all

--debug
    set log level to logging.DEBUG (maximize logging output)
    Equivalent to: [--Application.log_level=10]
--show-config
    Show the application's configuration (human-readable format)
    Equivalent to: [--Application.show_config=True]
--show-config-json
    Show the application's configuration (json format)
    Equivalent to: [--Application.show_config_json=True]
--generate-config
    generate default config file
    Equivalent to: [--JupyterApp.generate_config=True]
-y
    Answer yes to any questions instead of prompting.
    Equivalent to: [--JupyterApp.answer_yes=True]
--execute
    Execute the notebook prior to export.
    Equivalent to: [--ExecutePreprocessor.enabled=True]
--allow-errors
    Continue notebook execution even if one of the cells throws an error and
include the error message in the cell output (the default behaviour is to abort
conversion). This flag is only relevant if '--execute' was specified, too.
    Equivalent to: [--ExecutePreprocessor.allow_errors=True]
--stdin
    read a single notebook file from stdin. Write the resulting notebook with
default basename 'notebook.*'
    Equivalent to: [--NbConvertApp.from_stdin=True]
--stdout
    Write notebook output to stdout instead of files.
    Equivalent to: [--NbConvertApp.writer_class=StdoutWriter]
--inplace
    Run nbconvert in place, overwriting the existing notebook (only
            relevant when converting to notebook format)
    Equivalent to: [--NbConvertApp.use_output_suffix=False
--NbConvertApp.export_format=notebook --FilesWriter.build_directory=]
--clear-output
```

```
    Clear output of current file and save in place,
            overwriting the existing notebook.
    Equivalent to: [--NbConvertApp.use_output_suffix=False
--NbConvertApp.export_format=notebook --FilesWriter.build_directory=
--ClearOutputPreprocessor.enabled=True]
--coalesce-streams
    Coalesce consecutive stdout and stderr outputs into one stream (within each
cell).
    Equivalent to: [--NbConvertApp.use_output_suffix=False
--NbConvertApp.export_format=notebook --FilesWriter.build_directory=
--CoalesceStreamsPreprocessor.enabled=True]
--no-prompt
    Exclude input and output prompts from converted document.
    Equivalent to: [--TemplateExporter.exclude_input_prompt=True
--TemplateExporter.exclude_output_prompt=True]
--no-input
    Exclude input cells and output prompts from converted document.
            This mode is ideal for generating code-free reports.
    Equivalent to: [--TemplateExporter.exclude_output_prompt=True
--TemplateExporter.exclude_input=True
--TemplateExporter.exclude_input_prompt=True]
--allow-chromium-download
    Whether to allow downloading chromium if no suitable version is found on the
system.
    Equivalent to: [--WebPDFExporter.allow_chromium_download=True]
--disable-chromium-sandbox
    Disable chromium security sandbox when converting to PDF..
    Equivalent to: [--WebPDFExporter.disable_sandbox=True]
--show-input
    Shows code input. This flag is only useful for dejavu users.
    Equivalent to: [--TemplateExporter.exclude_input=False]
--embed-images
    Embed the images as base64 dataurls in the output. This flag is only useful
for the HTML/WebPDF/Slides exports.
    Equivalent to: [--HTMLExporter.embed_images=True]
--sanitize-html
    Whether the HTML in Markdown cells and cell outputs should be sanitized..
    Equivalent to: [--HTMLExporter.sanitize_html=True]
--log-level=<Enum>
    Set the log level by value or name.
    Choices: any of [0, 10, 20, 30, 40, 50, 'DEBUG', 'INFO', 'WARN', 'ERROR',
'CRITICAL']
    Default: 30
    Equivalent to: [--Application.log_level]
--config=<Unicode>
    Full path of a config file.
    Default: ''
    Equivalent to: [--JupyterApp.config_file]
```

```
--to=<Unicode>
    The export format to be used, either one of the built-in formats
            ['asciidoc', 'custom', 'html', 'latex', 'markdown', 'notebook',
'pdf', 'python', 'qtpdf', 'qtpng', 'rst', 'script', 'slides', 'webpdf']
            or a dotted object name that represents the import path for an
            ``Exporter`` class
    Default: ''
    Equivalent to: [--NbConvertApp.export_format]
--template=<Unicode>
    Name of the template to use
    Default: ''
    Equivalent to: [--TemplateExporter.template_name]
--template-file=<Unicode>
    Name of the template file to use
    Default: None
    Equivalent to: [--TemplateExporter.template_file]
--theme=<Unicode>
    Template specific theme(e.g. the name of a JupyterLab CSS theme distributed
    as prebuilt extension for the lab template)
    Default: 'light'
    Equivalent to: [--HTMLExporter.theme]
--sanitize_html=<Bool>
    Whether the HTML in Markdown cells and cell outputs should be sanitized.This
    should be set to True by nbviewer or similar tools.
    Default: False
    Equivalent to: [--HTMLExporter.sanitize_html]
--writer=<DottedObjectName>
    Writer class used to write the
                                        results of the conversion
    Default: 'FilesWriter'
    Equivalent to: [--NbConvertApp.writer_class]
--post=<DottedOrNone>
    PostProcessor class used to write the
                                        results of the conversion
    Default: ''
    Equivalent to: [--NbConvertApp.postprocessor_class]
--output=<Unicode>
    Overwrite base name use for output files.
                Supports pattern replacements '{notebook_name}'.
    Default: '{notebook_name}'
    Equivalent to: [--NbConvertApp.output_base]
--output-dir=<Unicode>
    Directory to write output(s) to. Defaults
                                    to output to the directory of each notebook.
To recover
                                    previous default behaviour (outputting to the
current
                                    working directory) use . as the flag value.
```

```
    Default: ''
    Equivalent to: [--FilesWriter.build_directory]
--reveal-prefix=<Unicode>
    The URL prefix for reveal.js (version 3.x).
            This defaults to the reveal CDN, but can be any url pointing to a
copy
            of reveal.js.
            For speaker notes to work, this must be a relative path to a local
            copy of reveal.js: e.g., "reveal.js".
            If a relative path is given, it must be a subdirectory of the
            current directory (from which the server is run).
            See the usage documentation
            (https://nbconvert.readthedocs.io/en/latest/usage.html#reveal-js-
html-slideshow)
            for more details.
    Default: ''
    Equivalent to: [--SlidesExporter.reveal_url_prefix]
--nbformat=<Enum>
    The nbformat version to write.
            Use this to downgrade notebooks.
    Choices: any of [1, 2, 3, 4]
    Default: 4
    Equivalent to: [--NotebookExporter.nbformat_version]

Examples
--------

    The simplest way to use nbconvert is

            > jupyter nbconvert mynotebook.ipynb --to html

            Options include ['asciidoc', 'custom', 'html', 'latex', 'markdown',
'notebook', 'pdf', 'python', 'qtpdf', 'qtpng', 'rst', 'script', 'slides',
'webpdf'].

            > jupyter nbconvert --to latex mynotebook.ipynb

            Both HTML and LaTeX support multiple output templates. LaTeX
includes
            'base', 'article' and 'report'.  HTML includes 'basic', 'lab' and
            'classic'. You can specify the flavor of the format used.

            > jupyter nbconvert --to html --template lab mynotebook.ipynb

            You can also pipe the output to stdout, rather than a file

            > jupyter nbconvert mynotebook.ipynb --stdout
```

```
            PDF is generated via latex

            > jupyter nbconvert mynotebook.ipynb --to pdf

            You can get (and serve) a Reveal.js-powered slideshow

            > jupyter nbconvert myslides.ipynb --to slides --post serve

            Multiple notebooks can be given at the command line in a couple of
            different ways:

            > jupyter nbconvert notebook*.ipynb
            > jupyter nbconvert notebook1.ipynb notebook2.ipynb

            or you can specify the notebooks list in a config file, containing::

                c.NbConvertApp.notebooks = ["my_notebook.ipynb"]

            > jupyter nbconvert --config mycfg.py

To see all available configurables, use `--help-all`.
```