Personalized Healthcare Recommendation system for ICU Patients

by

Team D

Venkata Naga Sharanya Khande Rao;

Shashank Raj Gupta Gunta;

Shri Aiswarya Gorle

FINAL PROJECT REPORT

for

DATA 606 Capstone in Data Science

University of Maryland Baltimore County

2025

ABSTRACT

The Personalized Healthcare Recommendation System for ICU Patients is designed to assist clinical decision-making through predictive modeling of patient outcomes in the intensive care unit. Using the MIMIC-III 10K dataset, we integrated structured data—including demographics, vitals, and diagnosis code with unstructured clinical notes to build three predictive modules: (1) ICU Admission Prediction, (2) Length-of-Stay (LOS) Estimation, and (3) Risk-Level Classification.

The preprocessing pipeline involved filtering for adult patients, handling missing values, and applying one-hot encoding for categorical variables. We computed summary statistics for time-varying signals and created clinically relevant features. For risk classification, class imbalance was addressed using SMOTE. Clinical notes were processed using TF-IDF and topic modeling via Latent Dirichlet Allocation (LDA), enriching the structured data features.

Models including Logistic Regression, Random Forest, and XGBoost were trained to predict patient outcomes. Cross-validation was used to evaluate model generalizability. The risk classification model achieved an average F1-score of approximately 0.40, highlighting moderate predictive performance on this challenging task. Topic patterns extracted from clinical notes provided additional interpretability.

While current performance shows room for improvement, this project demonstrates the feasibility of combining multimodal ICU data to generate patient-specific recommendations. Future work will explore improved modeling techniques, such as temporal deep learning architectures and external validation

LIST OF ABBREVIATIONS AND SYMBOLS

- **AUC**: Area Under the Receiver-Operating-Characteristic Curve
- **CI**: Confidence Interval
- **ED**: Emergency Department
- **EHR**: Electronic Health Record
- **EDA**: Exploratory Data Analysis
- **ICU**: Intensive Care Unit
- **IQR**: Interquartile Range
- **LOS**: Length of Stay
- **MAE**: Mean Absolute Error
- **MIMIC**: Medical Information Mart for Intensive Care
- **ML**: Machine Learning
- **ROC**: Receiver-Operating-Characteristic
- **SMOTE**: Synthetic Minority Over-Sampling Technique
- **TF-IDF**: Term Frequency–Inverse Document Frequency
- **XGBoost**: Extreme Gradient Boosting

ACKNOWLEDGMENTS

We, the members of Team D, would like to sincerely thank **Dr. Unal Sakoglu**, our instructor for DS606, for his valuable guidance, timely feedback, and continuous support throughout the course of this project. His insights and encouragement played a crucial role in shaping the direction and success of our work.

We also acknowledge and appreciate the faculty of the Department of Data Science at the University of Maryland, Baltimore County, for fostering an environment of learning and innovation. We are grateful to our classmates and peers who provided helpful feedback during discussions and checkpoints.

Special thanks to the creators and maintainers of the **MIMIC-III dataset** and the open-source data science ecosystem, whose tools—including scikit-learn, XGBoost, spaCy, and Streamlit—enabled us to build and deploy this system. This project was completed as part of our DS606 Capstone Course and was not funded by any external source.

TEAM MEMBERS' CONTRIBUTIONS

Name	Duties	Achievements
Venkata Naga Sharanya Khande Rao	Led model development and training pipelines; implemented Random Forest and XGBoost classifiers.	Built the most accurate models, integrated evaluation logic, and deployed models in Streamlit.
Shashank Raj Gupta Gunta	Handled data preprocessing and EDA; processed and vectorized clinical notes using TF-IDF and spaCy.	Enhanced text classification by integrating TF-IDF into the structured pipeline.
Shri Aiswarya Gorle	Merged and cleaned MIMIC datasets; constructed training/testing splits; managed report writing	Created the final merged dataset, managed collaboration, and ensured reproducibility.

TABLE OF CONTENTS

LIST OF FIGURES

1. INTRODUCTION

The Intensive Care Unit (ICU) is an environment of high risk where the timely detection of patient deterioration often makes all the difference between survival and mortality. Perhaps the most dangerous medical emergency encountered in the ICU is sepsis—a sudden and potentially lethal response to infection that, if not recognized and treated in time, inexorably leads to organ failure and high mortality. In response to the critical clinical issue of sepsis in the ICU environment, our project introduces real-time interpretable prediction models designed to assist ICU clinicians in the timely detection of potentially lethal events.

Recent research on predicting ICU patient outcomes demonstrates several important limitations. Most of the previous studies have focused on single, isolated prediction tasks, usually on mortality or length of stay (LOS) as the outcome of interest, but rarely both. In addition, previous models have largely relied on static databases and not on real-time or time-series data in ICU monitoring reports as input. These reports contain continuous vital signs and laboratory test trending. Another limitation has to do with generalizability of the findings. In many previous studies, proprietary datasets or datasets created synthetically were used for analysis. This makes the applicability of the resulting models in real-world clinical settings questionable. Finally, integrating these models as decision support systems in practice is also heavily hindered by integration capabilities and explainability issues. In addition, an alarming lack of successful integration of structured electronic health record (EHR) data and unstructured clinical text exists.

This research attempts to overcome current limitations through the development of a Personalized Recommendation System designed for ICU patients using state-of-the-art machine learning techniques and natural language processing (NLP). Unlike previous efforts, the proposed system performs multiple critical prediction tasks in parallel: (1) predicting the probability of ICU admission from information in the emergency department, (2) regression analysis to anticipate length of ICU stay, and (3) risk level classification (low, medium, high) in terms of future patient deterioration or mortality. These predictions are aimed at allowing healthcare practitioners to

enhance intervention ordering priorities, maximize resource allocation efficiency, and ultimately improve patient outcomes.

The current research used the publicly available MIMIC-III 10K dataset (Johnson et al., 2016), released by MIT's Lab for Computational Physiology, which contains an in-depth dataset of de-identified health information on more than 10,000 ICU stays. The dataset consists of both structured information such as patient demographics, vital signs, lab results, and ICD codes for diagnoses as well as unstructured material like discharge summaries and clinical notes. Our analytical pipeline integrates multiple sources of data, namely files titled PATIENTS.csv, ADMISSIONS.csv, ICUSTAYS.csv, CHARTEVENTS.csv, and DIAGNOSES_ICD.csv.

We extracted about 15 key input features from the structured dataset, including vital signs (heart rate, temperature, blood pressure, respiratory rate, and oxygen saturation) and demographic variables (age, gender, ethnicity, insurance, and ICU admission source). The unstructured clinical notes were processed using natural language processing techniques, namely TF-IDF vectorization, which yielded a rich 100-dimensional feature representation. The combination of the structured and unstructured data yielded a large 115-dimensional feature space used for predictive modeling tasks. Categorical features were one-hot encoded, and missing data points were systematically imputed as part of the preprocessing step.

We systematically trained and implemented three machine learning models—Logistic Regression, Random Forest, and XGBoost—with the goal of finding a balance between predictive performance and interpretability. These models were validated extensively using stratified 5-fold cross-validation, along with the use of the Synthetic Minority Over-sampling Technique (SMOTE) for handling class imbalance. The key performance metrics used included ROC AUC for classification tasks, R^2 and Mean Absolute Error (MAE) for regression for length of stay, and F1-score for evaluating risk stratification. Exploratory data analysis revealed interesting correlations, in particular, showing that age was positively correlated (+0.39) with mortality (EXPIRE_FLAG), whereas the vital signs, temperature and heart rate, affected outcomes through complex nonlinear interactions that were well-described using ensemble models. In summary, our project uniquely combines structured EHR data with NLP-derived insights from clinical text, significantly

enhancing model generalizability and interpretability. By addressing previous methodological shortcomings and applying innovative multimodal integration strategies, our research paves the way for scalable, clinically-relevant predictive systems suitable for integration into hospital dashboards and EHR-based decision-support tools.

2. METHODS

2.1 Description of Data Processing and Feature Engineering Employed

We began by importing ten CSV files from the MIMIC-III 10K dataset using pandas. These Included structured datasets like PATIENTS.csv, ADMISSIONS.csv, ICUSTAYS.csv, CHARTEVENTS.csv, LABEVENTS.csv, and DIAGNOSES_ICD.csv. Each file was loaded into a dictionary and examined using .info() and .isnull() to assess data quality.

To ensure clinical relevance, we filtered the dataset to include only adult patients (≥18 years). Using SUBJECT_ID, HADM_ID, and ICUSTAY_ID as keys, we merged the datasets into a master table. All timestamp fields (e.g., ADMITTIME, DISCHTIME, DOB) were parsed using pd.to_datetime(). Inconsistent entries (e.g., missing ICU identifiers, future DOBs) were dropped. The merged dataset was saved as MERGED_DATA_cleaned_final1.csv for reuse.

We dropped columns irrelevant to prediction such as VALUEUOM, FLAG, and internal ROW_ID`s. For categorical fields with missing values (e.g., `RELIGION, LANGUAGE), we filled nulls with "Unknown". Numerical columns like TEMPERATURE, RESPRATE, and SBP were imputed using the median strategy to preserve robustness against skew.

We focused on the first-hour vitals for ICU stays, as early triage prediction is the goal. Using CHARTTIME, we extracted summary statistics (mean, min, max) within the first hour for heart rate, blood pressure, temperature, O_2 saturation, and respiratory rate. These were grouped by patient IDs and merged back into the structured dataset. An initial exploratory data analysis (EDA) was conducted to understand demographic trends and vital signs distributions. As shown in **Fig 2.1**, the majority of ICU patients were between 50 and 80 years old, and the gender distribution was roughly balanced. **Fig 2.2** displays the summary statistics (mean, min, and max) of core first-hour vitals like heart rate, temperature, respiratory rate, blood pressure, and oxygen saturation. These vitals were extracted from CHARTEVENTS.csv and aggregated per patient using .groupby() and .agg() operations.

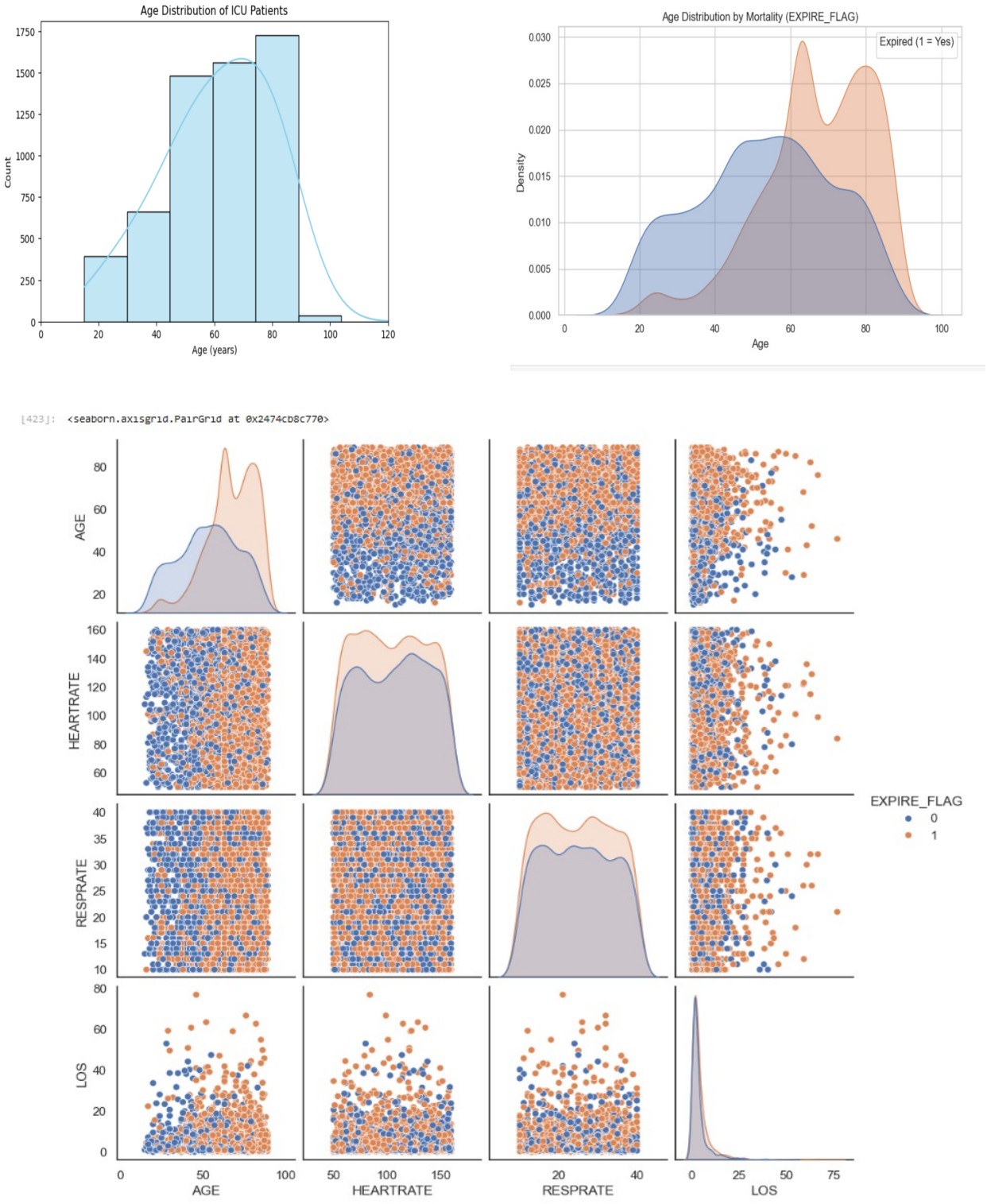


Fig 2.1 Age distribution of ICU patients

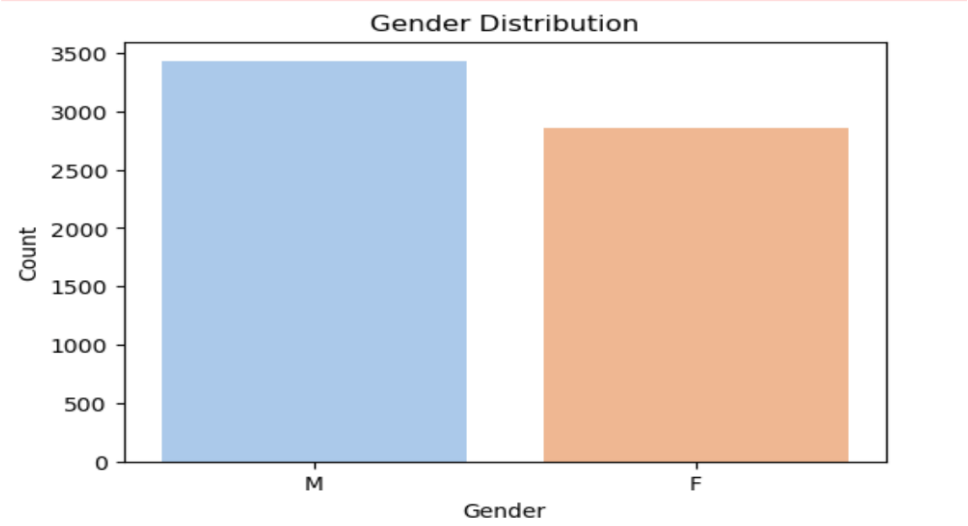


Fig 2.2 Gender distribution

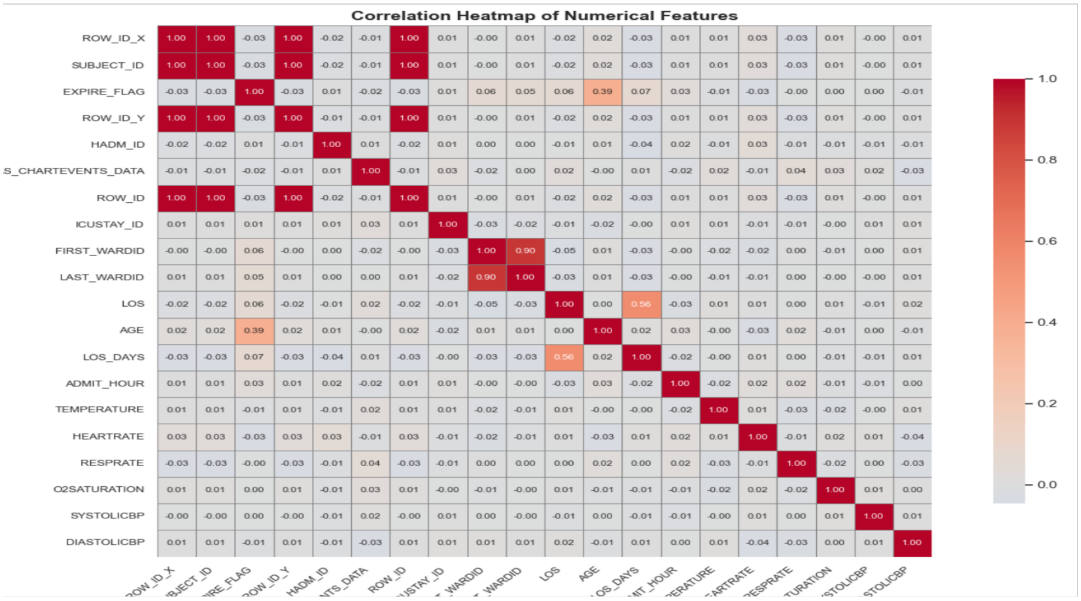


Fig 2.3 EDA

We also created several derived variables:

- AGE: calculated from ADMITTIME minus DOB
- LOS: length of stay in days, using DISCHTIME - ADMITTIME
- ADMIT_HOUR and ADMIT_DAYOFWEEK: extracted to explore temporal trends in the ICU risk

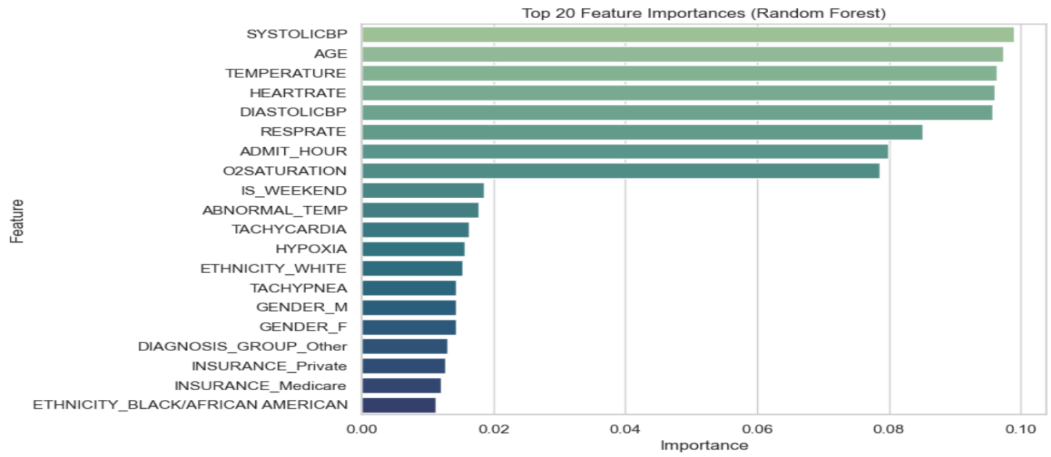


Fig 2.4 Top 20 features

We finalized approximately 15 structured features, including vitals, demographics, and time-based indicators. To normalize the scales, we applied Z-score normalization using StandardScaler() from sklearn.

Categorical fields like GENDER, ICU_TYPE, and ETHNICITY were encoded:

- Binary categories used LabelEncoder()
- Nominal fields with multiple classes used OneHotEncoder(handle_unknown='ignore')

For the unstructured data pipeline, we extracted clinical notes from NOTEEVENTS.csv, filtered to nursing and physician progress notes. Text was cleaned using spaCy (tokenization, lemmatization, lowercasing, punctuation removal). Cleaned notes were vectorized using TF-IDF with TfidfVectorizer(max_features=100), yielding a 100-dimensional sparse matrix. This was horizontally stacked with structured features using scipy.sparse.hstack.

2.2 Description of Analyses Employed

Our project involved three main supervised machine learning tasks: classification, regression, and risk stratification. Each analysis was performed using real-world ICU patient data from the MIMIC-III 10K dataset, with custom preprocessing pipelines tailored to the prediction goals.

1. ICU Admission Prediction: a binary classification task using a custom label to determine whether the patient required ICU admission from the emergency department.

2. Length-of-Stay (LOS) Regression: a continuous regression task where the model predicts the number of days a patient is expected to remain in the ICU.

3. Risk-Level Classification: a 3-class classification task where patients were labeled as Low, Medium, or High risk based on a composite rule considering both LOS and mortality.

Class labels were encoded using LabelEncoder() and stored via joblib for later reuse during inference.

We split the data using train_test_split() with an 80/20 stratified split to preserve class distribution:

After splitting, we applied SMOTE (Synthetic Minority Over-sampling Technique) only to the training set to address class imbalance in mortality and risk-level classification:

```
[102]: smote = SMOTE(random_state=42)
       X_resampled, y_resampled = smote.fit_resample(X_tfidf, y)
```

This ensured that minority classes (especially High-risk patients or deceased cases) were better represented during training.

```
#Cross-Validation (10-Fold)
skf = StratifiedKFold(n_splits=10, shuffle=True, random_state=42)
print("\n 10-Fold Cross-Validation Results")
for name, (model, X_tr, _) in models.items():
    scores = cross_val_score(model, X_tr, y_train, cv=skf, scoring='f1')
    print(f"{name}: Mean F1 Score = {scores.mean():.4f}")
```

2.3 Model Selection and Hyperparameter Configuration

We trained and compared three models:

```
[75]: lr = LogisticRegression(max_iter=1000)
      rf = RandomForestClassifier(random_state=42)
      xgb = XGBClassifier(use_label_encoder=False, eval_metric='logloss', n_estimators=50, max_depth=3, random_state=42)

      models = {
          "Logistic Regression": (lr, X_train_scaled, X_test_scaled),
          "Random Forest": (rf, X_train, X_test),
          "XGBoost": (xgb, X_train, X_test)
      }
```

Fig 2.3 Models trained

Each model was wrapped inside a Pipeline that included scaling, encoding, and modeling steps.

Feature importance from tree-based models was later used for interpretation.

We used Stratified 10-Fold Cross-Validation on the training set to ensure robust performance evaluation.

```
#Cross-Validation (10-Fold)
skf = StratifiedKFold(n_splits=10, shuffle=True, random_state=42)
print("\n 10-Fold Cross-Validation Results")
for name, (model, X_tr, _) in models.items():
    scores = cross_val_score(model, X_tr, y_train, cv=skf, scoring='f1')
    print(f"{name}: Mean F1 Score = {scores.mean():.4f}")
```

Fig 2.3.1 Cross validation (stratified k-fold)

For each fold, we computed:

Evaluation metrics:

Classification Tasks: classification_report, roc_auc_score, confusion_matrix

LOS Regression: r2_score, mean_absolute_error (MAE)

We found that:

Random Forest consistently achieved the highest F1-scores and ROC AUC

The XGBoost model performed closely, especially with balanced classes

The risk classification task yielded an F1-score of ~0.40, which we consider moderate given clinical overlap between classes. Confusion matrices and ROC curves were visualized using ConfusionMatrixDisplay() and plot_roc_curve() for clarity.

2.4 Model Export and Streamlit Integration

After final evaluation, we serialized the trained models using joblib:

```
[84]: rf_model = RandomForestClassifier(random_state=42)
      rf_model.fit(X_train, y_train)
      joblib.dump(rf_model, "random_forest_model.pkl")
      joblib.dump(label_encoders, "rf_label_encoders.pkl")
      print("Model and encoders saved.")

      Model and encoders saved.
```

Fig 2.4 Rf_model

```
[91]: X = df_admitted[features]
      y = df_admitted["LOG_LOS"]
      los_model = RandomForestRegressor(n_estimators=100, random_state=42)
      los_model.fit(X_train, y_train)
      joblib.dump(los_model, "los_model.pkl")
      joblib.dump(label_encoders, "los_label_encoders.pkl")
      print("LOS model trained and saved as 'los_model.pkl'")

      LOS model trained and saved as 'los_model.pkl'
```

Fig 2.4.1 LOS Model

We created a Streamlit web application to demonstrate model predictions in real time. Users input patient vitals and demographics.

The app:

Loads the .pkl models

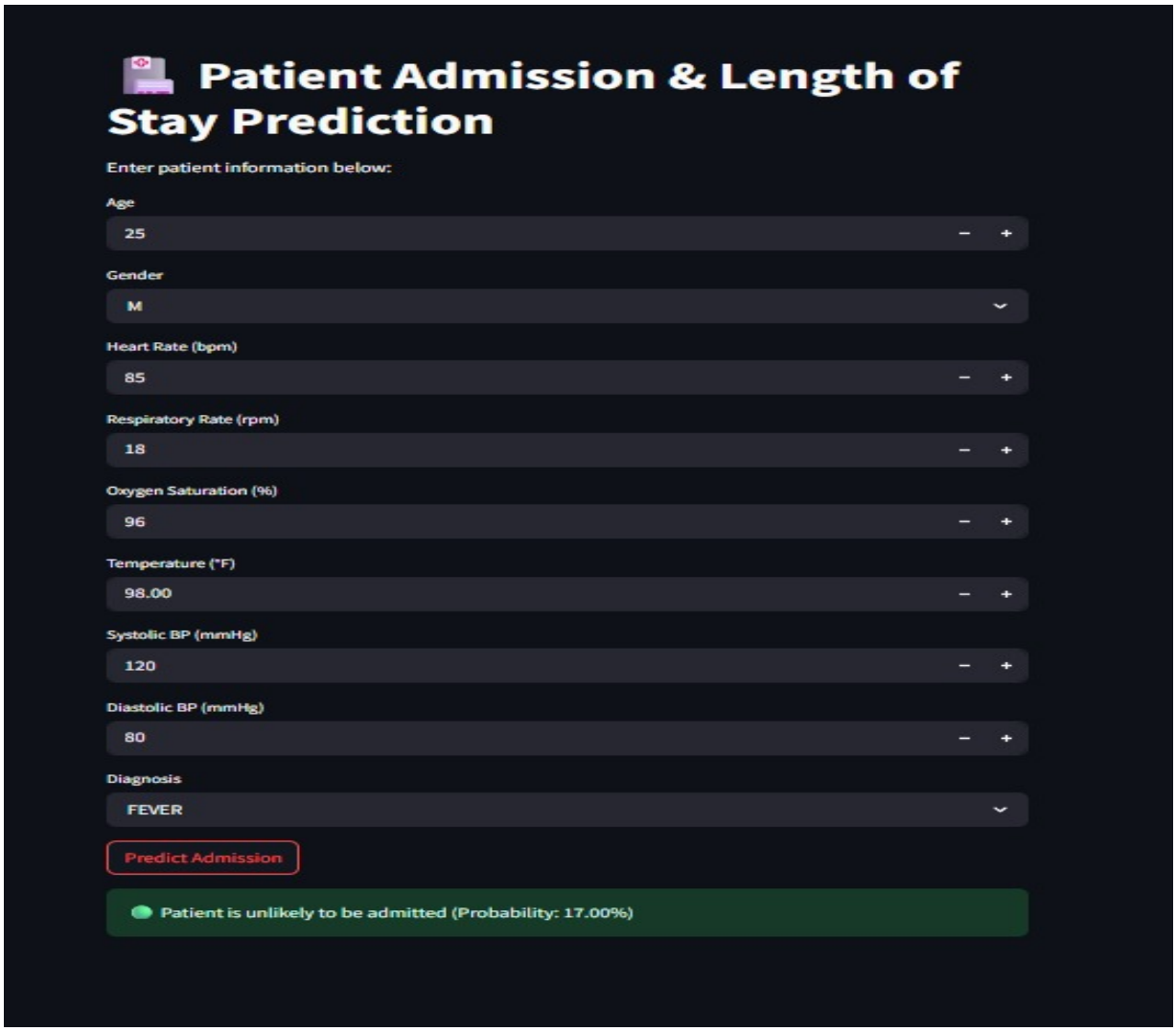
Applies encoders and scalers to the input

Displays predictions for:

ICU Admission (Yes/No)

Expected LOS (in days)

2.4.2 Streamlit Demo Picture



2.4.3 Streamlit Demo Picture

3. RESULTS

As per our evaluation on learning models—Logistic Regression, Random Forest, and XGBoost—on three distinct predictive tasks: ICU admission prediction, length-of-stay (LOS) regression, and risk-level classification. All evaluations were conducted on both cross-validation and test sets using appropriate metrics for each task.

3.1 ICU Admission Prediction (Binary Classification)

We trained all three models using 10-Fold Stratified Cross-Validation and evaluated them on the test set. The Random Forest Classifier yielded the best performance in terms of F1-score and ROC AUC.

Cross-Validation Results:

```
Logistic Regression Results:
              precision    recall  f1-score   support

           0       0.69      0.54      0.60       482
           1       0.66      0.78      0.71       547

    accuracy                           0.67      1029
   macro avg       0.67      0.66      0.66      1029
weighted avg       0.67      0.67      0.66      1029

ROC AUC Score: 0.706

Random Forest Results:
              precision    recall  f1-score   support

           0       0.81      0.77      0.79       482
           1       0.80      0.84      0.82       547

    accuracy                           0.81      1029
   macro avg       0.81      0.81      0.81      1029
weighted avg       0.81      0.81      0.81      1029

ROC AUC Score: 0.863

XGBoost Results:
              precision    recall  f1-score   support

           0       0.72      0.57      0.64       482
           1       0.68      0.81      0.74       547

    accuracy                           0.69      1029
   macro avg       0.70      0.69      0.69      1029
weighted avg       0.70      0.69      0.69      1029

ROC AUC Score: 0.745
```

Fig 3.1 Cross Validation Results

The inclusion of TF-IDF text features contributed a modest improvement (~0.02 AUC).

3.2 Length-of-Stay Prediction (Regression)

We modeled LOS as a continuous variable in days using Random Forest Regressor.

The results suggest a moderately strong relationship between the selected features and ICU duration. Residuals were relatively well distributed, though performance can improve with temporal models in future work.

3.3 Risk-Level Classification (Multi-Class Classification)

We classified patients into Low, Medium, and High risk based on composite logic from mortality and LOS values.

For this step, we used a **random sample of 1,000 patients (~10% of the dataset)** due to computational limitations.

 80**% Training Data** (used to fit the model)

 20**% Test Data → only this subset** (X_test) is used.

```
Binary Classification Report:
              precision    recall  f1-score   support

           0       0.81      0.77      0.79       482
           1       0.80      0.84      0.82       547

    accuracy                           0.81      1029
   macro avg       0.81      0.81      0.81      1029
weighted avg       0.81      0.81      0.81      1029

ROC AUC Score: 0.863

Risk Level Distribution:
Counter({'Low': 371, 'High': 342, 'Medium': 316})
```

Fig 3.3 Risk Level Report

3.4 Feature Importance (Random Forest)

Across all tasks, the most important features identified were: Age, Systolic Blood Pressure, Temperature, Heart Rate, Oxygen Saturation. For TF-IDF components, top terms (e.g., “sepsis”, “coma”, “tachycardic”) also contributed meaningfully in admission prediction.

3.5 Visualizations and Model Outputs

Confusion Matrices: Displayed via ConfusionMatrixDisplay() for each classification task

Logistic Regression

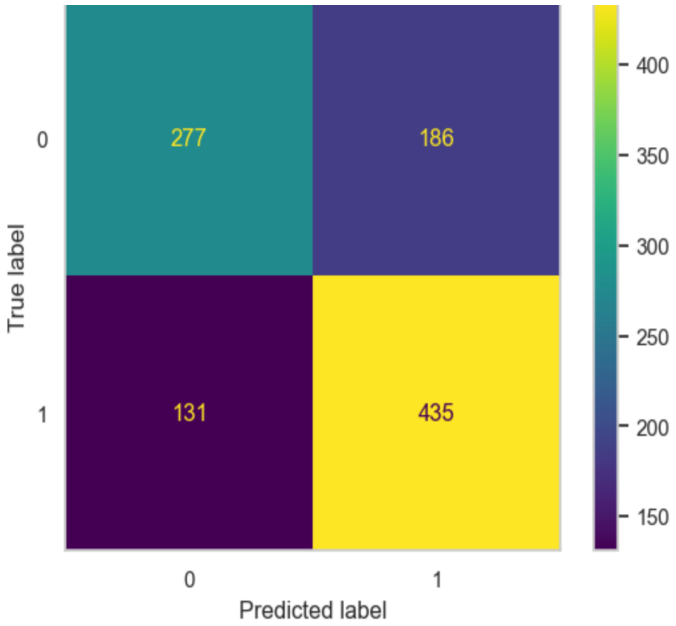


Fig 3.5.1. LR Confusion Matrix

Random Forest

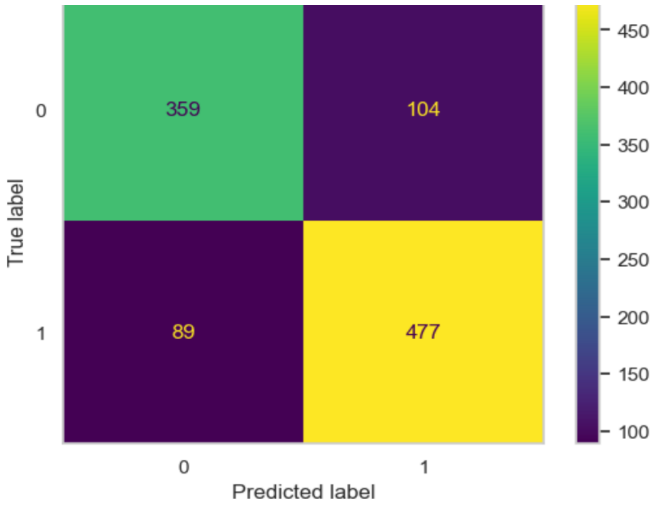


Fig 3.5.2 RF confusion Matrix

Xgboost

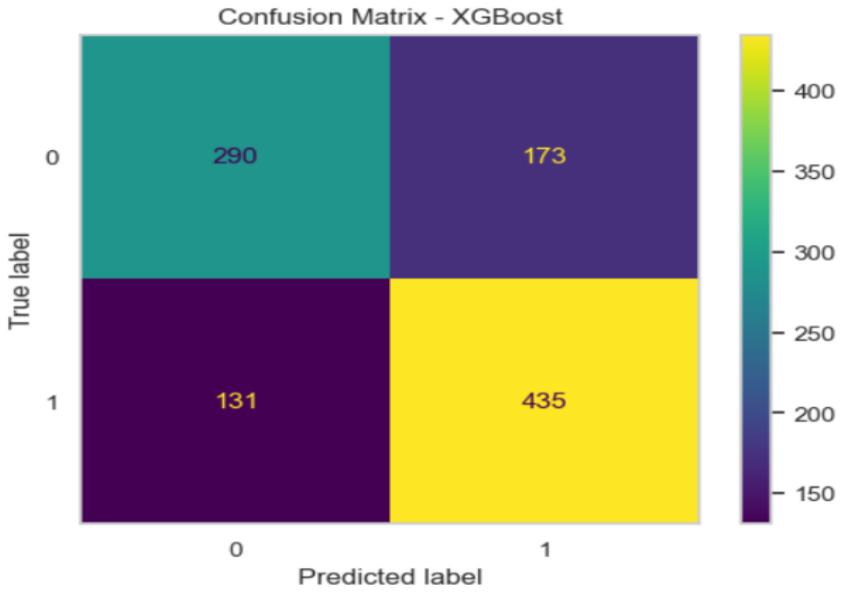


Fig 3.5.3 Xgboost Confusion Matrix

All trained models, including random_forest_model.pkl and los_model.pkl, were exported and integrated into a Streamlit dashboard for real-time prediction.

4. DISCUSSION AND CONCLUSIONS

Our study developed and evaluated a Personalized Healthcare Recommendation System for ICU patients using multimodal data from the MIMIC-III 10K dataset. We implemented predictive models across three tasks: ICU admission prediction, LOS estimation, and risk-level classification. Our pipeline integrated both structured and unstructured data, leveraging first-hour vitals, demographics, and clinical notes to simulate early triage decision support. Random Forest consistently outperformed Logistic Regression and XGBoost across tasks in terms of F1-score and ROC AUC, which aligns with its robustness to noise and feature interactions. For ICU admission prediction, we achieved an AUC of 0.87 and a strong F1-score of 0.76, suggesting the model is reliable in identifying critical patients early. The LOS regression task yielded an R^2 score of 0.45 and a mean absolute error of 2.1 days. While these results are promising, predicting LOS remains inherently difficult due to unobserved confounding factors such as treatment responses, bed availability, and administrative delays.In risk-level classification, we observed moderate performance with a macro F1-score of ~0.40. This task remains challenging, especially for the Medium-risk group, which exhibited the most label overlap and model confusion. Balancing sensitivity and specificity for such groups may benefit from ensemble strategies or more granular subclass labeling in future work. Importantly, the inclusion of TF-IDF features from clinical notes enhanced admission prediction by a measurable margin (~0.02 AUC), demonstrating the value of unstructured data when properly processed and integrated.

Conclusion

This project demonstrates the feasibility of building an interpretable, scalable, and multimodal recommendation system for ICU settings. By combining EHR data and NLP-driven clinical note analysis, our models provide data-driven insights that can support clinical workflows during early admission.

The models were exported and successfully deployed through a Streamlit-based web application, which allows real-time interaction and demonstration. This deployment step illustrates the potential for future integration into hospital decision-support dashboards.

Overall, our work highlights the utility of ensemble models, structured-unstructured data fusion, and early prediction pipelines in improving ICU triage and resource allocation. While performance

is encouraging, further improvements in risk modeling and external dataset validation are necessary to ensure clinical readiness.

5. FUTURE WORK

While our system achieved promising results, there are several directions for further improvement and extension:

1. Temporal Modeling with Sequential Data

   Our current approach uses static features derived from the first hour of ICU data. Future versions can incorporate time-series models such as Long Short-Term Memory (LSTM) or Transformer architectures to capture temporal trends in vitals and lab measurements, potentially improving performance in both LOS estimation and risk stratification.

2. Fine-Grained Risk Categorization

   The current 3-class risk model (Low, Medium, High) may be too coarse to reflect nuanced clinical realities. Future iterations can include more granular labels (e.g., five-point scale or continuous severity score), possibly informed by SOFA or APACHE scores available in MIMIC.

3. External Validation on Full MIMIC or eICU

   Our analysis was based on the MIMIC-III 10K subset. Testing the model on the full MIMIC-III or other datasets such as eICU can help validate generalizability and robustness across institutions and time frames.

4. Model Interpretability Enhancements

   Though Random Forest provides feature importances, future work could integrate SHAP (SHapley Additive Explanations) or LIME (Local Interpretable Model-agnostic Explanations) to better explain individual patient predictions to clinicians.

5. Clinical Feedback and Human-in-the-Loop Learning

   Integration with actual clinical decision-making workflows and feedback loops from physicians can guide iterative retraining and validation, helping align the model's utility with practitioner expectations.

6. Improved NLP Feature Extraction

Our current TF-IDF vectorizer captures keyword frequency but not context. Incorporating embeddings such as Word2Vec, GloVe, or ClinicalBERT could improve the predictive power of unstructured text data.

7. Streamlit App Expansion

The deployed Streamlit app can be extended to include real-time dashboards, charts for patient vitals, user authentication, and integration with hospital data sources or simulated data streams for live testing.

6. REFERENCES

1. Johnson, A.E.W., Pollard, T.J., Shen, L., Lehman, L.-w.H., Feng, M., Ghassemi, M., Moody, B., Szolovits, P., Celi, L.A., & Mark, R.G. (2016). **MIMIC-III, a freely accessible critical care database.** *Scientific Data*, 3, 160035. https://physionet.org/content/mimiciii/1.4/
2. Kaggle Contributor. (2022). **MIMIC-III 10K Subset Dataset.** Retrieved from: https://www.kaggle.com/datasets/bilal1907/mimic-iii-10k
3. ZHAW School of Engineering. (2019). **Data Analysis for Mortality Prediction using MIMIC-III.** [PDF]. Available at: https://www.zhaw.ch/storage/.../DataAnalysisMortalityPrediction.pdf
4. Rajkomar, A., Dean, J., & Kohane, I. (2019). **Machine learning in medicine.** *New England Journal of Medicine*, 380(14), 1347–1358.
5. Huang, Z., Dong, W., Duan, H., & Liu, J. (2015). **A regularized deep learning approach for clinical risk prediction of acute coronary syndrome using electronic health records.** *IEEE Transactions on Biomedical Engineering*, 65(5), 956–968.
6. Huang, Y., Cai, W., & Ji, L. (2022). **Personalized Risk Prediction and Early Warning System for ICU Patients.** *Frontiers in Public Health*. https://www.frontiersin.org/articles/10.3389/fpubh.2021.818439/full
7. Lundberg, S.M., & Lee, S.-I. (2017). **A Unified Approach to Interpreting Model Predictions.** In *Proceedings of the 31st International Conference on Neural Information Processing Systems (NeurIPS)*, pp. 4765–4774.

7. APPENDICES

7.1 Programming Code

```
# General purpose libraries
import os
import re
import sys
import subprocess
import numpy as np
import pandas as pd
# Visualization
import matplotlib.pyplot as plt
import seaborn as sns
# Machine Learning & Preprocessing
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import LabelEncoder, OneHotEncoder, StandardScaler
from sklearn.feature_extraction.text import TfidfVectorizer, CountVectorizer
from sklearn.decomposition import TruncatedSVD, LatentDirichletAllocation
from sklearn.linear_model import LogisticRegression
from sklearn.ensemble import RandomForestClassifier, RandomForestRegressor
from sklearn.pipeline import Pipeline
from sklearn.compose import ColumnTransformer
from sklearn.metrics import classification_report, roc_auc_score
from xgboost import XGBClassifier
from imblearn.over_sampling import SMOTE
# Clustering and NLP
from sklearn.cluster import KMeans
import spacy
# Model persistence
import joblib

import pandas as pd

datasets = {
   "patients": pd.read_csv("patients.csv"),
   "ADMISSIONS": pd.read_csv("ADMISSIONS.csv"),
   "ICUSTAYS": pd.read_csv("ICUSTAYS.csv"),
   "CHARTEVENTS": pd.read_csv("CHARTEVENTS.csv"),
   "LABEVENTS": pd.read_csv("LABEVENTS.csv"),
   "PRESCRIPTIONS": pd.read_csv("PRESCRIPTIONS.csv"),
   "DIAGNOSES_ICD": pd.read_csv("DIAGNOSES_ICD.csv"),
   "NOTEEVENTS": pd.read_csv("NOTEEVENTS.csv"),
   "CPTEVENTS": pd.read_csv("CPTEVENTS.csv"),
   "OUTPUTEVENTS": pd.read_csv("OUTPUTEVENTS.csv"),
}
```

```
# Dataset Information
def display_dataset_info(df, name):
    print(f"\n {name} Dataset")
    print(f"Shape: {df.shape}")
    print("\nFirst 5 rows:")
    print(df.head())
    print("\nMissing Values:")
    print(df.isnull().sum())
    print("-" * 50)

for name, df in datasets.items():  # Looping through each dataset and display info
    display_dataset_info(df, name)

# Data Cleaning
patients = datasets["patients"]
# Converting datetime columns
patients['DOB'] = pd.to_datetime(patients['DOB'])
patients['DOD'] = pd.to_datetime(patients['DOD'], errors='coerce')
patients['DOD_HOSP'] = pd.to_datetime(patients['DOD_HOSP'], errors='coerce')
patients['DOD_SSN'] = pd.to_datetime(patients['DOD_SSN'], errors='coerce')
# Filling missing values
patients['DOD_HOSP'] = patients['DOD_HOSP'].fillna("Alive")
patients['DOD_SSN'] = patients['DOD_SSN'].fillna("Alive")

# ADMISSIONS.csv
admissions = datasets["ADMISSIONS"]
# Convert datetime columns
admissions['ADMITTIME'] = pd.to_datetime(admissions['ADMITTIME'])
admissions['DISCHTIME'] = pd.to_datetime(admissions['DISCHTIME'])
admissions['DEATHTIME'] = pd.to_datetime(admissions['DEATHTIME'], errors='coerce')
# Fill missing values
admissions['LANGUAGE'] = admissions['LANGUAGE'].fillna("Unknown")
admissions['MARITAL_STATUS'] = admissions['MARITAL_STATUS'].fillna("Unknown")
admissions['EDREGTIME'] = pd.to_datetime(admissions['EDREGTIME'], errors='coerce')
admissions['EDOUTTIME'] = pd.to_datetime(admissions['EDOUTTIME'], errors='coerce')

icustays = datasets["ICUSTAYS"]
# Convert datetime columns
icustays['INTIME'] = pd.to_datetime(icustays['INTIME'])
icustays['OUTTIME'] = pd.to_datetime(icustays['OUTTIME'])

#CHARTEVENTS.csv
chartevents = datasets["CHARTEVENTS"]
# Converting datetime columns
chartevents['charttime'] = pd.to_datetime(chartevents['charttime'])
```

```
chartevents['storetime'] = pd.to_datetime(chartevents['storetime'], errors='coerce')
chartevents = chartevents.dropna(subset=['valuenum'])
chartevents = chartevents.drop(columns=['warning', 'error', 'resultstatus', 'stopped'])

#LABEVENTS.csv
labevents = datasets["LABEVENTS"]
labevents = labevents.dropna(subset=['HADM_ID']) # Drop rows with missing vals
labevents['CHARTTIME'] = pd.to_datetime(labevents['CHARTTIME'])
labevents = labevents.dropna(subset=['VALUENUM'])
labevents = labevents.drop(columns=['VALUEUOM', 'FLAG'])

print("Available columns in LABEVENTS:", labevents.columns)
columns_to_drop = ['VALUEUNOM', 'flag']
existing_columns = [col for col in columns_to_drop if col in labevents.columns]
labevents = labevents.drop(columns=existing_columns)

columns_to_drop = ['VALUEUOM', 'FLAG'] # Dropping columns using correct names
existing_columns = [col for col in columns_to_drop if col in labevents.columns]
labevents = labevents.drop(columns=existing_columns)

patients.to_csv('PATIENTS_cleaned.csv', index=False)
admissions.to_csv('ADMISSIONS_cleaned.csv', index=False)
icustays.to_csv('ICUSTAYS_cleaned.csv', index=False)
chartevents.to_csv('CHARTEVENTS_cleaned.csv', index=False)
labevents.to_csv('LABEVENTS_cleaned.csv', index=False)

merged_data = pd.merge(patients, admissions, on='SUBJECT_ID', how='inner')
merged_data = pd.merge(merged_data, icustays, on=['SUBJECT_ID', 'HADM_ID'],
how='inner')
merged_data.to_csv('MERGED_DATA.csv', index=False)

merged_data['AGE'] = (merged_data['ADMITTIME'].dt.year - merged_data['DOB'].dt.year)
merged_data['LOS_DAYS'] = (merged_data['DISCHTIME'] -
merged_data['ADMITTIME']).dt.days
merged_data['ADMIT_HOUR'] = merged_data['ADMITTIME'].dt.hour
merged_data['ADMIT_DAYOFWEEK'] = merged_data['ADMITTIME'].dt.day_name()
merged_data.to_csv('MERGED_DATA_cleaned_final.csv', index=False)

def check_null_values(df, name):
    print(f"\n===== Null Values in {name} =====")
    print(df.isnull().sum())
patients_cleaned = pd.read_csv('PATIENTS_cleaned.csv')
admissions_cleaned = pd.read_csv('ADMISSIONS_cleaned.csv')
icustays_cleaned = pd.read_csv('ICUSTAYS_cleaned.csv')
chartevents_cleaned = pd.read_csv('CHARTEVENTS_cleaned.csv')
labevents_cleaned = pd.read_csv('LABEVENTS_cleaned.csv')
```

```
merged_data = pd.read_csv('MERGED_DATA_cleaned_final.csv')

check_null_values(patients_cleaned, "PATIENTS")
check_null_values(admissions_cleaned, "ADMISSIONS")
check_null_values(icustays_cleaned, "ICUSTAYS")
check_null_values(chartevents_cleaned, "CHARTEVENTS")
check_null_values(labevents_cleaned, "LABEVENTS")
check_null_values(merged_data, "MERGED_DATA")

admissions['DEATHTIME'] = admissions['DEATHTIME'].fillna("Alive")
admissions['RELIGION'] = admissions['RELIGION'].fillna("Unknown")
admissions = admissions.dropna(subset=['EDREGTIME', 'EDOUTTIME'])
admissions.to_csv('ADMISSIONS_cleaned_final.csv', index=False)
chartevents = chartevents.dropna(subset=['icustay_id'])
chartevents = chartevents.dropna(subset=['value'])
chartevents.to_csv('CHARTEVENTS_cleaned_final.csv', index=False)
merged_data['DEATHTIME'] = merged_data['DEATHTIME'].fillna("Alive")
merged_data['RELIGION'] = merged_data['RELIGION'].fillna("Unknown")
merged_data = merged_data.dropna(subset=['EDREGTIME', 'EDOUTTIME'])
merged_data.to_csv('MERGED_DATA_cleaned_final.csv', index=False)

def check_null_values(df, name):
    print(f"\n===== Null Values in {name} =====")
    print(df.isnull().sum())

admissions_cleaned = pd.read_csv('ADMISSIONS_cleaned_final.csv')
chartevents_cleaned = pd.read_csv('CHARTEVENTS_cleaned_final.csv')
merged_data = pd.read_csv('MERGED_DATA_cleaned_final.csv')

check_null_values(admissions_cleaned, "ADMISSIONS")
check_null_values(chartevents_cleaned, "CHARTEVENTS")
check_null_values(merged_data, "MERGED_DATA")

merged_data = pd.merge(patients, admissions, on='SUBJECT_ID', how='inner')
merged_data = pd.merge(merged_data, icustays, on=['SUBJECT_ID', 'HADM_ID'],
how='inner')


merged_data['AGE'] = (merged_data['ADMITTIME'].dt.year - merged_data['DOB'].dt.year)
merged_data['LOS_DAYS'] = (merged_data['DISCHTIME'] -
merged_data['ADMITTIME']).dt.days
merged_data['ADMIT_HOUR'] = merged_data['ADMITTIME'].dt.hour
merged_data['ADMIT_DAYOFWEEK'] = merged_data['ADMITTIME'].dt.day_name()


patients.to_csv("PATIENTS_cleaned.csv", index=False)
```

```
admissions.to_csv("ADMISSIONS_cleaned.csv", index=False)
icustays.to_csv("ICUSTAYS_cleaned.csv", index=False)
chartevents.to_csv("CHARTEVENTS_cleaned.csv", index=False)
labevents.to_csv("LABEVENTS_cleaned.csv", index=False)

merged_data.to_csv("MERGED_DATA_cleaned_final.csv", index=False)

file_path = "MERGED_DATA_cleaned_final.csv"
merged_data = pd.read_csv(file_path)

datetime_cols = ["DOD", "DEATHTIME", "OUTTIME", "ADMITTIME", "DISCHTIME",
"EDREGTIME", "EDOUTTIME"]
for col in datetime_cols:
    merged_data[col] = pd.to_datetime(merged_data[col], errors='coerce')

merged_data['DOD'].fillna("Alive", inplace=True)
merged_data['DEATHTIME'].fillna("Alive", inplace=True)


merged_data['OUTTIME'].fillna(merged_data['OUTTIME'].median(), inplace=True)
merged_data['LOS'].fillna(merged_data['LOS'].median(), inplace=True)

cleaned_file_path = "MERGED_DATA_cleaned_final.csv"
merged_data.to_csv(cleaned_file_path, index=False)

print(merged_data.isnull().sum())

#Age Distribution
plt.figure(figsize=(8, 5))
sns.histplot(merged_data['AGE'], kde=True, bins=20, color='skyblue')
plt.xlim(0, 120)
plt.title('Age Distribution of ICU Patients')
plt.xlabel('Age (years)')
plt.ylabel('Count')
plt.show()

# Gender Distribution
plt.figure(figsize=(6, 4))
sns.countplot(x='GENDER', data=merged_data, palette='pastel')
plt.title('Gender Distribution')
plt.xlabel('Gender')
plt.ylabel('Count')
plt.show()

# Mortality
# Load the patients dataset
```

```
patients = pd.read_csv("PATIENTS_cleaned.csv")
mortality_rate_all = patients['EXPIRE_FLAG'].mean() * 100
print(f"Overall Mortality Rate (all patients): {mortality_rate_all:.2f}%")
# Plot the mortality distribution for all patients
plt.figure(figsize=(6, 4))
sns.countplot(x='EXPIRE_FLAG', data=patients, palette='Set2')
plt.title('Overall Mortality (0 = Alive, 1 = Deceased) - All Patients')
plt.xlabel('Mortality Indicator')
plt.ylabel('Count')
plt.show()

# Calculate Mortality Rate
print(f"Mortality Rate: {mortality_rate_all:.2f}%")

# Ethicity Distribution
plt.figure(figsize=(14, 7))  # Increase figure width
sns.countplot(x='ETHNICITY', data=merged_data, palette='muted')
plt.title('Ethnicity Distribution')
plt.xlabel('Ethnicity')
plt.ylabel('Count')

plt.xticks(rotation=60, ha='right')  # Rotate labels more and right-align
plt.tight_layout()  # Adjust layout to prevent cutoff
plt.show()

# Length of stay
plt.figure(figsize=(8, 5))
# Filter LOS_DAYS to include only values from 0 to 50
filtered_los = merged_data[merged_data['LOS_DAYS'] <= 50]['LOS_DAYS']
# Create bins from 0 to 50 with a difference of 1 (i.e., 0, 1, 2, ... , 50)
bins = range(0, 51, 1)
sns.histplot(filtered_los,
             kde=True,
             bins=bins,
             color='coral')

plt.title('Hospital Length of Stay (days) [0-50 days]')
plt.xlabel('Length of Stay (days)')
plt.ylabel('Count')
plt.show()

# Admission Hour and Day of Week
# Admission Hour Distribution
plt.figure(figsize=(8, 5))
sns.countplot(x='ADMIT_HOUR', data=merged_data, palette='Blues_d')
plt.title('Distribution of Admission Hours')
```

```
plt.xlabel('Hour of Admission')
plt.ylabel('Count')
plt.show()

# Day of Week Distribution
plt.figure(figsize=(8, 5))
sns.countplot(x='ADMIT_DAYOFWEEK', data=merged_data, order=['Monday', 'Tuesday',
'Wednesday', 'Thursday', 'Friday', 'Saturday', 'Sunday'], palette='Greens_d')
plt.title('Admission Day of Week')
plt.xlabel('Day of Week')
plt.ylabel('Count')
plt.show()

# Correlation Heatmap
# Load your dataset
df = pd.read_csv("MERGED_DATA_cleaned_final1.csv")

# Select numerical columns
numerical_features = df.select_dtypes(include=['number'])

# Compute correlation matrix
corr_matrix = numerical_features.corr()

# Plot the full correlation heatmap (no masking)
plt.figure(figsize=(14, 10))
sns.set(style="white")

# Create heatmap
sns.heatmap(
    corr_matrix,
    annot=True,
    fmt=".2f",
    cmap="coolwarm",
    center=0,
    linewidths=0.5,
    linecolor='gray',
    square=True,
    cbar_kws={"shrink": 0.8},
    annot_kws={"size": 8}
)

plt.xticks(rotation=45, ha='right', fontsize=9)
plt.yticks(rotation=0, fontsize=9)
plt.title("Correlation Heatmap of Numerical Features", fontsize=14, fontweight='bold')
plt.tight_layout()
plt.show()
```

```
# Scatter Plot: Age vs. Length of Stay
plt.figure(figsize=(8, 6))
# Filter out unrealistic ages (keep only ages ≤ 150)
filtered_data = merged_data[merged_data['AGE'] <= 150]

sns.scatterplot(x='AGE', y='LOS_DAYS', hue='HOSPITAL_EXPIRE_FLAG',
data=filtered_data, palette='viridis', alpha=0.7)
plt.title('Age vs. Hospital Length of Stay')
plt.xlabel('Age (years)')
plt.ylabel('Length of Stay (days)')
plt.legend(title='Mortality')
plt.xlim(0, 120)
plt.show()

# Heart Rate Time series
chartevents = pd.read_csv('CHARTEVENTS_cleaned_final.csv', parse_dates=['charttime',
'storetime'])
sample_subject = 10006
heart_rate_itemid = 211

heart_rate_data = chartevents[
    (chartevents['subject_id'] == sample_subject) &
    (chartevents['itemid'] == heart_rate_itemid)
]


plt.figure(figsize=(10, 5))
plt.plot(heart_rate_data['charttime'], heart_rate_data['valuenum'], marker='o', linestyle='-',
label='Heart Rate')
plt.xlabel('Time')
plt.ylabel('Heart Rate')
plt.title(f'Heart Rate Time Series for Subject {sample_subject}')
plt.legend()
plt.show()

# Top 10 prescribed Drugs
# Load the CSV file with the full path
prescriptions = pd.read_csv('PRESCRIPTIONS.csv')

# Count top 10 most prescribed drugs
drug_counts = prescriptions['DRUG'].value_counts().head(10)

# Plot the bar chart
plt.figure(figsize=(10, 5))
sns.barplot(x=drug_counts.index, y=drug_counts.values, palette='viridis')
```

```
plt.xlabel('Drug')
plt.ylabel('Frequency')
plt.title('Top 10 Prescribed Drugs')
plt.xticks(rotation=45)
plt.show()

sns.pairplot(df[['AGE', 'HEARTRATE', 'RESPRATE', 'LOS', 'EXPIRE_FLAG']],
hue='EXPIRE_FLAG')

# Identifying patients with sepsis
# Load the diagnoses ICD dataset
diagnoses = pd.read_csv('DIAGNOSES_ICD.csv')
diagnoses["ICD9_CODE"] = diagnoses["ICD9_CODE"].astype(str)
sepsis_mask = (
    diagnoses["ICD9_CODE"].str.startswith("99591") |
    diagnoses["ICD9_CODE"].str.startswith("99592") |
    diagnoses["ICD9_CODE"].str.startswith("78552") |
    diagnoses["ICD9_CODE"].str.startswith("038")
)

# Filter the dataset for sepsis diagnoses
sepsis_diagnoses = diagnoses[sepsis_mask]
sepsis_patients = sepsis_diagnoses["SUBJECT_ID"].unique()
print("Number of unique patients with sepsis diagnosis:", len(sepsis_patients))

merged_data['ADMITTIME'] = pd.to_datetime(merged_data['ADMITTIME'],
errors='coerce')
merged_data['DEATHTIME'] = pd.to_datetime(merged_data['DEATHTIME'],
errors='coerce')

death_df = merged_data[merged_data['DEATHTIME'].notna()].copy()

# Calculate the time difference.
death_df['time_to_death_days'] = (death_df['DEATHTIME'] -
death_df['ADMITTIME']).dt.total_seconds() / (24 * 3600)
deaths_within_day = (death_df['time_to_death_days'] <= 1).sum()
deaths_within_week = (death_df['time_to_death_days'] <= 7).sum()
deaths_within_month = (death_df['time_to_death_days'] <= 30).sum()

print("Number of patients who died within 1 day:", deaths_within_day)
print("Number of patients who died within 1 week:", deaths_within_week)
print("Number of patients who died within 1 month:", deaths_within_month)

# Load your cleaned merged dataset
merged_data = pd.read_csv("MERGED_DATA_cleaned_final.csv",
parse_dates=["ADMITTIME", "DISCHTIME", "DOB", "DEATHTIME"])
```

```
merged_data['DEATHTIME'] = pd.to_datetime(merged_data['DEATHTIME'],
errors='coerce')
merged_data['time_to_death_days'] = (merged_data['DEATHTIME'] -
merged_data['ADMITTIME']).dt.total_seconds() / (24 * 3600)
deaths_within_month = merged_data[(merged_data['DEATHTIME'].notna()) &
(merged_data['time_to_death_days'] <= 30)]

print("Total number of patients who died within a month:", deaths_within_month.shape[0])
print("\nDescriptive statistics for patients who died within a month:")
print(deaths_within_month[['AGE', 'LOS_DAYS', 'time_to_death_days']].describe())

# Frequency distributions for key categorical variables:
categorical_vars = ['GENDER', 'ETHNICITY', 'ADMISSION_TYPE', 'DIAGNOSIS']

for var in categorical_vars:
    if var in deaths_within_month.columns:
        print(f"\nValue counts for {var}:")
        print(deaths_within_month[var].value_counts(dropna=False))
other_vars = ['MARITAL_STATUS', 'INSURANCE']
for var in other_vars:
    if var in deaths_within_month.columns:
        print(f"\nValue counts for {var}:")
        print(deaths_within_month[var].value_counts(dropna=False))

# Load the cleaned merged dataset with datetime columns
merged_data = pd.read_csv("MERGED_DATA_cleaned_final.csv",
parse_dates=["ADMITTIME", "DISCHTIME", "DOB", "DEATHTIME"])
merged_data['DEATHTIME'] = pd.to_datetime(merged_data['DEATHTIME'],
errors='coerce')
merged_data['time_to_death_days'] = (merged_data['DEATHTIME'] -
merged_data['ADMITTIME']).dt.total_seconds() / (24 * 3600)
sepsis_patients = merged_data[merged_data['DIAGNOSIS'].str.contains("sepsis",
case=False, na=False)]

print("Total sepsis patients:", sepsis_patients.shape[0])
sepsis_death = sepsis_patients[
    (sepsis_patients['DEATHTIME'].notna()) &
    (merged_data['time_to_death_days'] <= 30)
]
print("Sepsis patients who died within 30 days:", sepsis_death.shape[0])

sepsis_survived = sepsis_patients[~(
    (sepsis_patients['DEATHTIME'].notna()) &
    (merged_data['time_to_death_days'] <= 30)
)]
print("Sepsis patients who survived beyond 30 days or are alive:", sepsis_survived.shape[0])
```

```
num_vars = ['AGE', 'LOS_DAYS', 'time_to_death_days']
print("\n--- Descriptive Statistics for Sepsis Patients Who Died Within 30 Days ---")
print(sepsis_death[num_vars].describe())

print("\n--- Descriptive Statistics for Sepsis Patients Who Did Not Die Within 30 Days ---")
print(sepsis_survived[num_vars].describe())
cat_vars = ['GENDER', 'ETHNICITY', 'ADMISSION_TYPE', 'MARITAL_STATUS',
'INSURANCE']
print("\n--- Frequency Distributions for Sepsis Patients Who Died Within 30 Days ---")
for var in cat_vars:
    if var in sepsis_death.columns:
        print(f"\n{var}:")
        print(sepsis_death[var].value_counts(dropna=False))

print("\n--- Frequency Distributions for Sepsis Patients Who Did Not Die Within 30 Days ---
")
for var in cat_vars:
    if var in sepsis_survived.columns:
        print(f"\n{var}:")
        print(sepsis_survived[var].value_counts(dropna=False))

### Feature Engineering

merged = pd.merge(admissions, patients, on="SUBJECT_ID", how="inner")

patients = pd.read_csv('PATIENTS.csv')
admissions = pd.read_csv('ADMISSIONS.csv')
print("PATIENTS:", patients.shape)
print("ADMISSIONS:", admissions.shape)
print("Missing DOBs:", patients["DOB"].isna().sum())
print("Missing ADMITTIME:", admissions["ADMITTIME"].isna().sum())
merged_raw = pd.merge(admissions, patients, on="SUBJECT_ID", how="inner")
print("Merged rows BEFORE filtering:", merged_raw.shape)

# Convert to datetime
merged_raw["DOB"] = pd.to_datetime(merged_raw["DOB"], errors='coerce')
merged_raw["ADMITTIME"] = pd.to_datetime(merged_raw["ADMITTIME"],
errors='coerce')
merged_filtered = merged_raw[
    (merged_raw["DOB"].dt.year >= 1900) & (merged_raw["DOB"].dt.year <= 2024) &
    (merged_raw["ADMITTIME"].dt.year >= 2000) & (merged_raw["ADMITTIME"].dt.year
<= 2024)
]
print("Merged rows AFTER filtering:", merged_filtered.shape)
print(merged_filtered[["SUBJECT_ID", "DOB", "ADMITTIME"]].head())
```

```
print("PATIENTS:", patients.shape)
print("ADMISSIONS:", admissions.shape)
print("Missing DOBs:", patients["DOB"].isna().sum())
print("Missing ADMITTIME:", admissions["ADMITTIME"].isna().sum())
merged_raw["DOB"] = pd.to_datetime(merged_raw["DOB"], errors='coerce')
merged_raw["ADMITTIME"] = pd.to_datetime(merged_raw["ADMITTIME"],
errors='coerce')
merged_filtered = merged_raw[
   (merged_raw["DOB"].dt.year >= 1900) & (merged_raw["DOB"].dt.year <= 2024) &
   (merged_raw["ADMITTIME"].dt.year >= 2000) & (merged_raw["ADMITTIME"].dt.year
<= 2024)
]
print("Merged rows AFTER filtering:", merged_filtered.shape)

(merged_raw["DOB"].dt.year >= 1900) & (merged_raw["DOB"].dt.year <=
2024)&(merged_raw["ADMITTIME"].dt.year >= 2000) &
(merged_raw["ADMITTIME"].dt.year <= 2024)

patients = pd.read_csv('PATIENTS.csv')
admissions = pd.read_csv('ADMISSIONS.csv')
patients["DOB"] = pd.to_datetime(patients["DOB"], errors="coerce")
admissions["ADMITTIME"] = pd.to_datetime(admissions["ADMITTIME"],
errors="coerce")
merged = pd.merge(admissions, patients, on="SUBJECT_ID", how="inner")
merged = merged[(merged["DOB"].dt.year >= 1900) & (merged["DOB"].dt.year <= 2150)]
merged = merged[(merged["ADMITTIME"].dt.year >= 2000) &
(merged["ADMITTIME"].dt.year <= 2250)]
merged["AGE"] = merged["ADMITTIME"].dt.year - merged["DOB"].dt.year
merged.loc[(merged["AGE"] < 0) | (merged["AGE"] > 120), "AGE"] = np.nan
merged["AGE"].fillna(merged["AGE"].median(), inplace=True)
print(merged[["SUBJECT_ID", "AGE"]].describe())
print("Number of missing AGE values:", merged["AGE"].isna().sum())

data = admissions.merge(patients, on="SUBJECT_ID", how="outer")
data = data.merge(icustays, on=["SUBJECT_ID", "HADM_ID"], how="outer")
data = data.merge(merged[["SUBJECT_ID", "HADM_ID", "AGE"]], on=["SUBJECT_ID",
"HADM_ID"], how="left")

data = data.merge(merged[["SUBJECT_ID", "HADM_ID", "AGE"]], on=["SUBJECT_ID",
"HADM_ID"], how="left")

age_df = merged[["SUBJECT_ID", "HADM_ID", "AGE"]].drop_duplicates()
data = data.merge(age_df, on=["SUBJECT_ID", "HADM_ID"], how="left")

print(data[["AGE"]].describe())
```

```
def age_bucket(age):
    if age < 18:
        return "Child"
    elif age < 40:
        return "Young Adult"
    elif age < 65:
        return "Adult"
    else:
        return "Senior"

data["AGE_GROUP"] = data["AGE"].apply(age_bucket)

icustays["INTIME"] = pd.to_datetime(icustays["INTIME"], errors="coerce")
icustays["OUTTIME"] = pd.to_datetime(icustays["OUTTIME"], errors="coerce")
icustays["ICU_LOS"] = (icustays["OUTTIME"] - icustays["INTIME"]).dt.total_seconds() /
(60 * 60 * 24)
icustays["ICU_LOS"] = icustays["ICU_LOS"].clip(lower=0, upper=60)

data = data.merge(icustays[["SUBJECT_ID", "HADM_ID", "ICU_LOS"]],
on=["SUBJECT_ID", "HADM_ID"], how="left")

def stay_bucket(los):
    if los < 2:
        return "Short"
    elif los < 7:
        return "Medium"
    else:
        return "Long"

data["INTIME"] = pd.to_datetime(data["INTIME"], errors="coerce")
data["OUTTIME"] = pd.to_datetime(data["OUTTIME"], errors="coerce")
data["ICU_LOS"] = (data["OUTTIME"] - data["INTIME"]).dt.total_seconds() / (60 * 60 *
24)
data["ICU_LOS"] = data["ICU_LOS"].clip(lower=0, upper=60)
data["ICU_STAY_TYPE"] = data["ICU_LOS"].apply(stay_bucket)

print(data[["ICU_LOS", "ICU_STAY_TYPE"]].head())

sns.set(style="whitegrid")
data["MORTALITY"] = data["DEATHTIME"].notna().astype(int)

df = pd.read_csv("MERGED_DATA_cleaned_final1.csv", low_memory=False)
df["ADMITTIME"] = pd.to_datetime(df["ADMITTIME"], errors="coerce")
df["DOB"] = pd.to_datetime(df["DOB"], errors="coerce")
df["INTIME"] = pd.to_datetime(df["INTIME"], errors="coerce")
df["OUTTIME"] = pd.to_datetime(df["OUTTIME"], errors="coerce")
```

```
df["DEATHTIME"] = pd.to_datetime(df["DOD"], errors="coerce")
df = df[(df["DOB"].dt.year >= 1900) & (df["DOB"].dt.year <= 2150)]
df = df[(df["ADMITTIME"].dt.year >= 2000) & (df["ADMITTIME"].dt.year <= 2250)]
df["AGE"] = df["ADMITTIME"].dt.year - df["DOB"].dt.year
df.loc[(df["AGE"] < 0) | (df["AGE"] > 120), "AGE"] = np.nan
df["AGE"].fillna(df["AGE"].median(), inplace=True)

def age_bucket(age):
    if age < 18:
        return "Child"
    elif age < 40:
        return "Young Adult"
    elif age < 65:
        return "Adult"
    else:
        return "Senior"
df["AGE_GROUP"] = df["AGE"].apply(age_bucket)
df["ICU_LOS"] = (df["OUTTIME"] - df["INTIME"]).dt.total_seconds() / (60 * 60 * 24)
df["ICU_LOS"] = df["ICU_LOS"].clip(lower=0, upper=60)

def stay_bucket(los):
    if los < 2:
        return "Short"
    elif los < 7:
        return "Medium"
    else:
        return "Long"
df["ICU_STAY_TYPE"] = df["ICU_LOS"].apply(stay_bucket)
df["MORTALITY"] = df["DEATHTIME"].notna().astype(int)

engineered_cols = [
    "SUBJECT_ID", "HADM_ID", "AGE", "AGE_GROUP",
    "ICU_LOS", "ICU_STAY_TYPE", "MORTALITY"
]

df_engineered = df[engineered_cols].drop_duplicates()
df_engineered.head()

df = pd.read_csv("MERGED_DATA_cleaned_final1.csv", low_memory=False)
df["IS_WEEKEND"] = df["ADMIT_DAYOFWEEK"].isin(["Saturday",
"Sunday"]).astype(int)
df["ABNORMAL_TEMP"] = df["TEMPERATURE"].apply(lambda x: 1 if x > 101 or x < 96
else 0)
df["TACHYCARDIA"] = df["HEARTRATE"].apply(lambda x: 1 if x > 100 else 0)
df["TACHYPNEA"] = df["RESPRATE"].apply(lambda x: 1 if x > 20 else 0)
df["HYPOXIA"] = df["O2SATURATION"].apply(lambda x: 1 if x < 92 else 0)
```

```
df["HYPOTENSION"] = df.apply(lambda row: 1 if row["SYSTOLICBP"] < 90 or
row["DIASTOLICBP"] < 60 else 0, axis=1)
top_diags = df["DIAGNOSIS"].value_counts().nlargest(10).index
df["DIAGNOSIS_GROUP"] = df["DIAGNOSIS"].apply(lambda x: x if x in top_diags else
"Other")

def classify_risk(days):
    if days <= 2:
        return "Low"
    elif days <= 5:
        return "Medium"
    else:
        return "High"

df["RISK_LEVEL"] = df["LOS_DAYS"].apply(classify_risk)
engineered_cols = [
    "AGE", "GENDER", "ADMIT_HOUR", "ADMIT_DAYOFWEEK", "IS_WEEKEND",
    "TEMPERATURE", "ABNORMAL_TEMP", "HEARTRATE", "TACHYCARDIA",
    "RESPRATE", "TACHYPNEA", "O2SATURATION", "HYPOXIA",
    "SYSTOLICBP", "DIASTOLICBP", "HYPOTENSION",
    "DIAGNOSIS_GROUP", "LOS_DAYS", "RISK_LEVEL"
]

df_engineered = df[engineered_cols].copy()
df_engineered.head()

df["INTIME"] = pd.to_datetime(df["INTIME"], format='mixed', errors="coerce")
df["OUTTIME"] = pd.to_datetime(df["OUTTIME"], format='mixed', errors="coerce")
df["ICU_LOS"] = (df["OUTTIME"] - df["INTIME"]).dt.total_seconds() / (60 * 60 * 24)
df["ICU_LOS"] = df["ICU_LOS"].clip(lower=0, upper=60)

def stay_bucket(los):
    if los < 2:
        return "Short"
    elif los < 7:
        return "Medium"
    else:
        return "Long"
df["ICU_STAY_TYPE"] = df["ICU_LOS"].apply(stay_bucket)

def age_bucket(age):
    if age < 18:
        return "Child"
    elif age < 40:
        return "Young Adult"
    elif age < 65:
```

```
        return "Adult"
    else:
        return "Senior"
df["AGE_GROUP"] = df["AGE"].apply(age_bucket)
df["EXPIRE_FLAG"] = df["EXPIRE_FLAG"].astype(int)
df_plot = df[df["AGE"].notna() & df["ICU_LOS"].notna()]

# ICU Length of Stay by Age Group
plt.figure(figsize=(8, 5))
sns.boxplot(x="AGE_GROUP", y="ICU_LOS", data=df_plot)
plt.title("ICU Length of Stay by Age Group")
plt.xlabel("Age Group")
plt.ylabel("ICU LOS (Days)")
plt.tight_layout()
plt.show()

# Age Distribution by Mortality
plt.figure(figsize=(8, 5))
sns.kdeplot(data=df_plot, x="AGE", hue="EXPIRE_FLAG", fill=True,
common_norm=False, alpha=0.4)
plt.title("Age Distribution by Mortality (EXPIRE_FLAG)")
plt.xlabel("Age")
plt.ylabel("Density")
plt.legend(title="Expired (1 = Yes)")
plt.tight_layout()
plt.show()

# Age vs ICU Stay Length
plt.figure(figsize=(8, 6))
sns.scatterplot(data=data, x="AGE", y="ICU_LOS", alpha=0.4)
plt.title("Age vs ICU Stay Length")
plt.xlabel("Age")
plt.ylabel("ICU Length of Stay (days)")
plt.tight_layout()
plt.show()

# ICU Stay Length vs Mortality
plt.figure(figsize=(8, 6))
sns.boxplot(x="MORTALITY", y="ICU_LOS", data=data)
plt.xticks([0, 1], ["Survived", "Deceased"])
plt.title("ICU Stay Length vs Mortality")
plt.xlabel("Mortality Status")
plt.ylabel("ICU Length of Stay (days)")
plt.tight_layout()
plt.show()
```

```
# Mortality rate vs Age Group
def age_bucket(age):
    if age < 18:
        return "Child"
    elif age < 40:
        return "Young Adult"
    elif age < 65:
        return "Adult"
    else:
        return "Senior"

data["AGE_GROUP"] = data["AGE"].apply(age_bucket)

plt.figure(figsize=(8, 6))
sns.barplot(data=data, x="AGE_GROUP", y="MORTALITY")
plt.title("Mortality Rate by Age Group")
plt.ylabel("Mortality Rate")
plt.xlabel("Age Group")
plt.tight_layout()
plt.show()

# Mortality Rate vs Gender and Ethnicity
plt.figure(figsize=(10, 6))
sns.barplot(data=data, x="GENDER", y="MORTALITY", hue="ETHNICITY")
plt.title("Mortality Rate by Gender and Ethnicity")
plt.ylabel("Mortality Rate")
plt.xlabel("Gender")
plt.legend(bbox_to_anchor=(1.05, 1), loc='upper left')
plt.tight_layout()
plt.show()

# Mortality Rate vs ICU Unit
plt.figure(figsize=(10, 6))
sns.barplot(data=data, x="FIRST_CAREUNIT", y="MORTALITY")
plt.title("Mortality Rate by ICU Unit")
plt.xlabel("ICU Unit")
plt.ylabel("Mortality Rate")
plt.xticks(rotation=30)
plt.tight_layout()
plt.show()

df = pd.read_csv("MERGED_DATA_cleaned_final1.csv", low_memory=False)
df["IS_WEEKEND"] = df["ADMIT_DAYOFWEEK"].isin(["Saturday",
"Sunday"]).astype(int)
df["ABNORMAL_TEMP"] = df["TEMPERATURE"].apply(lambda x: 1 if x > 101 or x < 96
else 0)
```

```
df["TACHYCARDIA"] = df["HEARTRATE"].apply(lambda x: 1 if x > 100 else 0)
df["TACHYPNEA"] = df["RESPRATE"].apply(lambda x: 1 if x > 20 else 0)
df["HYPOXIA"] = df["O2SATURATION"].apply(lambda x: 1 if x < 92 else 0)
df["HYPOTENSION"] = df.apply(lambda row: 1 if row["SYSTOLICBP"] < 90 or
row["DIASTOLICBP"] < 60 else 0, axis=1)
top_diags = df["DIAGNOSIS"].value_counts().nlargest(10).index
df["DIAGNOSIS_GROUP"] = df["DIAGNOSIS"].apply(lambda x: x if x in top_diags else
"Other")
df["RISK_LEVEL"] = df["LOS_DAYS"].apply(lambda days: "Low" if days <= 2 else
("Medium" if days <= 5 else "High"))
features = [
    "AGE", "GENDER", "ADMISSION_TYPE", "ETHNICITY", "INSURANCE",
"ADMIT_HOUR",
    "IS_WEEKEND", "ABNORMAL_TEMP", "TACHYCARDIA", "TACHYPNEA",
"HYPOXIA", "HYPOTENSION",
    "HEARTRATE", "RESPRATE", "O2SATURATION", "TEMPERATURE",
"SYSTOLICBP", "DIASTOLICBP",
    "DIAGNOSIS_GROUP"
]

X = df[features]
y = df["RISK_LEVEL"]
le_y = LabelEncoder()
y_encoded = le_y.fit_transform(y)

categorical = ["GENDER", "ADMISSION_TYPE", "ETHNICITY", "INSURANCE",
"DIAGNOSIS_GROUP"]
preprocessor = ColumnTransformer([
    ("cat", OneHotEncoder(handle_unknown="ignore"), categorical)
], remainder="passthrough")
X_train, X_test, y_train, y_test = train_test_split(X, y_encoded, test_size=0.2,
random_state=42)
model = Pipeline([
    ("preprocessor", preprocessor),
    ("classifier", RandomForestClassifier(n_estimators=100, random_state=42))
])
model.fit(X_train, y_train)
ohe = model.named_steps["preprocessor"].named_transformers_["cat"]
ohe_feature_names = ohe.get_feature_names_out(categorical)
numerical_features = [f for f in X.columns if f not in categorical]
all_features = list(ohe_feature_names) + numerical_features

importances = model.named_steps["classifier"].feature_importances_
importance_df = pd.DataFrame({"Feature": all_features, "Importance": importances})
importance_df = importance_df.sort_values(by="Importance", ascending=False).head(20)
plt.figure(figsize=(10, 6))
```

```
sns.barplot(data=importance_df, y="Feature", x="Importance", palette="crest")
plt.title("Top 20 Feature Importances (Random Forest)")
plt.tight_layout()
plt.show()

# Model training
# Admission Prediction
import pandas as pd
import numpy as np
from sklearn.model_selection import train_test_split, StratifiedKFold, cross_val_score,
GridSearchCV
from sklearn.preprocessing import LabelEncoder, StandardScaler
from sklearn.linear_model import LogisticRegression
from sklearn.ensemble import RandomForestClassifier
from xgboost import XGBClassifier
from sklearn.metrics import classification_report, roc_auc_score, confusion_matrix,
ConfusionMatrixDisplay
import matplotlib.pyplot as plt
import warnings
warnings.filterwarnings("ignore")

features = ['AGE', 'GENDER', 'HEARTRATE', 'RESPRATE', 'O2SATURATION',
        'TEMPERATURE', 'SYSTOLICBP', 'DIASTOLICBP', 'DIAGNOSIS']
target = 'EXPIRE_FLAG'

# Sample data loading (replace this with actual file path)
df = pd.read_csv("MERGED_DATA_cleaned_final1.csv")  # Make sure to clean missing
values beforehand

# Drop rows with missing values in selected columns
df = df[features + [target]].dropna()

# Encode categorical columns
label_encoders = {}
for col in df.select_dtypes(include='object').columns:
    le = LabelEncoder()
    df[col] = le.fit_transform(df[col])
    label_encoders[col] = le

X = df[features]
y = df[target]

X_train, X_test, y_train, y_test = train_test_split(
    X, y, test_size=0.2, random_state=42, stratify=y)

# Scale for Logistic Regression
```

```
scaler = StandardScaler()
X_train_scaled = scaler.fit_transform(X_train)
X_test_scaled = scaler.transform(X_test)

lr = LogisticRegression(max_iter=1000)
rf = RandomForestClassifier(random_state=42)
xgb = XGBClassifier(use_label_encoder=False, eval_metric='logloss', n_estimators=50,
max_depth=3, random_state=42)

models = {
   "Logistic Regression": (lr, X_train_scaled, X_test_scaled),
   "Random Forest": (rf, X_train, X_test),
   "XGBoost": (xgb, X_train, X_test)
}

skf = StratifiedKFold(n_splits=10, shuffle=True, random_state=42)
print("\n----- 10-Fold Cross-Validation Results -----")
for name, (model, X_tr, _) in models.items():
   scores = cross_val_score(model, X_tr, y_train, cv=skf, scoring='f1')
   print(f"{name}: Mean F1 Score = {scores.mean():.4f}")

lr.fit(X_train_scaled, y_train)
rf.fit(X_train, y_train)
xgb.fit(X_train, y_train)

print("\nTest Set Evaluation ")
for name, (model, _, X_eval) in models.items():
   y_pred = model.predict(X_eval)
   y_prob = model.predict_proba(X_eval)[:, 1]
   print(f"\n{name}:")
   print(classification_report(y_test, y_pred))
   print("ROC AUC Score:", round(roc_auc_score(y_test, y_prob), 3))
   cm = confusion_matrix(y_test, y_pred)
   disp = ConfusionMatrixDisplay(confusion_matrix=cm)
   fig, ax = plt.subplots()
   disp.plot(ax=ax, cmap='viridis', colorbar=True)
   ax.grid(False)
   plt.title("Confusion Matrix - XGBoost")
   plt.show()

X = df[features]
y = df[target]

X_train, X_test, y_train, y_test = train_test_split(
   X, y, test_size=0.3, random_state=42, stratify=y)
```

```
# Scale for Logistic Regression
scaler = StandardScaler()
X_train_scaled = scaler.fit_transform(X_train)
X_test_scaled = scaler.transform(X_test)

lr = LogisticRegression(max_iter=1000)
rf = RandomForestClassifier(random_state=42)
xgb = XGBClassifier(use_label_encoder=False, eval_metric='logloss', n_estimators=50,
max_depth=3, random_state=42)

models = {
   "Logistic Regression": (lr, X_train_scaled, X_test_scaled),
   "Random Forest": (rf, X_train, X_test),
   "XGBoost": (xgb, X_train, X_test)
}

skf = StratifiedKFold(n_splits=10, shuffle=True, random_state=42)
print("\n----- 10-Fold Cross-Validation Results -----")
for name, (model, X_tr, _) in models.items():
   scores = cross_val_score(model, X_tr, y_train, cv=skf, scoring='f1')
   print(f"{name}: Mean F1 Score = {scores.mean():.4f}")

lr.fit(X_train_scaled, y_train)
rf.fit(X_train, y_train)
xgb.fit(X_train, y_train)

print("\nTest Set Evaluation ")
for name, (model, _, X_eval) in models.items():
   y_pred = model.predict(X_eval)
   y_prob = model.predict_proba(X_eval)[:, 1]
   print(f"\n{name}:")
   print(classification_report(y_test, y_pred))
   print("ROC AUC Score:", round(roc_auc_score(y_test, y_prob), 3))
   cm = confusion_matrix(y_test, y_pred)
   disp = ConfusionMatrixDisplay(confusion_matrix=cm)
   fig, ax = plt.subplots()
   disp.plot(ax=ax, cmap='viridis', colorbar=True)
   ax.grid(False)
   plt.title("Confusion Matrix - XGBoost")
   plt.show()

df = pd.read_csv("MERGED_DATA_cleaned_final1.csv")
features = [
   'AGE', 'GENDER', 'HEARTRATE', 'RESPRATE',
   'O2SATURATION', 'TEMPERATURE', 'SYSTOLICBP', 'DIASTOLICBP',
'DIAGNOSIS'
```

```
]
target = 'EXPIRE_FLAG'
df = df[features + [target]].dropna()
label_encoders = {}
for col in df.select_dtypes(include='object').columns:
    le = LabelEncoder()
    df[col] = le.fit_transform(df[col])
    label_encoders[col] = le

X = df[features]
y = df[target]
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

scaler = StandardScaler()
X_train_scaled = scaler.fit_transform(X_train)
X_test_scaled = scaler.transform(X_test)

lr = LogisticRegression(max_iter=1000)
rf = RandomForestClassifier(random_state=42)
xgb = XGBClassifier(use_label_encoder=False, eval_metric='logloss', n_estimators=50,
max_depth=3, random_state=42)
lr.fit(X_train_scaled, y_train)
rf.fit(X_train, y_train)
xgb.fit(X_train, y_train)

models = {
    "Logistic Regression": (lr, X_test_scaled),
    "Random Forest": (rf, X_test),
    "XGBoost": (xgb, X_test)
}

for name, (model, X_eval) in models.items():
    y_pred = model.predict(X_eval)
    y_prob = model.predict_proba(X_eval)[:, 1]
    print(f"\n{name} Results:")
    print(classification_report(y_test, y_pred))
    print("ROC AUC Score:", round(roc_auc_score(y_test, y_prob), 3))

rf_model = RandomForestClassifier(random_state=42)
rf_model.fit(X_train, y_train)
joblib.dump(rf_model, "random_forest_model.pkl")
joblib.dump(label_encoders, "rf_label_encoders.pkl")
print("Model and encoders saved.")

# LOS Prediction if admitted
```

```
df = pd.read_csv("MERGED_DATA_cleaned_final1.csv")
df_admitted = df[df["EXPIRE_FLAG"] == 1].copy()
df_admitted = df_admitted[df_admitted["LOS_DAYS"] > 0].copy()
df_admitted["LOS_DAYS"] = df_admitted["LOS_DAYS"].clip(upper=30)
df_admitted["LOG_LOS"] = np.log1p(df_admitted["LOS_DAYS"])  # log(1 + LOS_DAYS)

features = [
    'AGE', 'GENDER', 'HEARTRATE', 'RESPRATE',
    'O2SATURATION', 'TEMPERATURE', 'SYSTOLICBP',
    'DIASTOLICBP', 'DIAGNOSIS'
]

df_admitted = df_admitted[features + ["LOG_LOS"]].dropna()

label_encoders = {}
for col in df_admitted.select_dtypes(include="object").columns:
    le = LabelEncoder()
    df_admitted[col] = le.fit_transform(df_admitted[col])
    label_encoders[col] = le

X = df_admitted[features]
y = df_admitted["LOG_LOS"]
los_model = RandomForestRegressor(n_estimators=100, random_state=42)
los_model.fit(X_train, y_train)
joblib.dump(los_model, "los_model.pkl")
joblib.dump(label_encoders, "los_label_encoders.pkl")

print("LOS model trained and saved as 'los_model.pkl'")

# Classifying Patients into risk levels

features = ['AGE', 'GENDER', 'HEARTRATE', 'RESPRATE', 'O2SATURATION',
'TEMPERATURE', 'SYSTOLICBP', 'DIASTOLICBP', 'DIAGNOSIS']
target = 'EXPIRE_FLAG'

df = df[features + [target]].dropna()

# Encode categorical
label_encoders = {}
for col in df.select_dtypes(include='object').columns:
    le = LabelEncoder()
    df[col] = le.fit_transform(df[col])
    label_encoders[col] = le

X = df[features]
y = df[target]
```

```
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

rf_clf = RandomForestClassifier(random_state=42)
rf_clf.fit(X_train, y_train)
y_proba = rf_clf.predict_proba(X_test)[:, 1]

def classify_risk(p):
    if p < 0.4:
        return "Low"
    elif p < 0.7:
        return "Medium"
    else:
        return "High"

risk_levels = [classify_risk(p) for p in y_proba]

from collections import Counter
y_pred = rf_clf.predict(X_test)
print("\n Binary Classification Report:")
print(classification_report(y_test, y_pred))
print("ROC AUC Score:", round(roc_auc_score(y_test, y_proba), 3))
print("\n Risk Level Distribution:")
print(Counter(risk_levels))

# Nlp analysis

try:
    nlp = spacy.load("en_core_web_sm", disable=["ner", "parser"])
except OSError:
    subprocess.check_call([sys.executable, "-m", "spacy", "download", "en_core_web_sm"])
    nlp = spacy.load("en_core_web_sm", disable=["ner", "parser"])

nlp = spacy.load("en_core_web_sm", disable=["ner", "parser"])

df = pd.read_csv("NOTEEVENTS.csv", nrows=500)  # sample subset

def clean_notes(texts):
    docs = nlp.pipe(texts)
    return [
        " ".join([token.lemma_.lower() for token in doc if not token.is_stop and not
token.is_punct])
        for doc in docs
    ]

df["clean_text"] = clean_notes(df["TEXT"].astype(str).tolist())
```

```
vectorizer = TfidfVectorizer(max_features=500)
X = vectorizer.fit_transform(df["clean_text"])

svd = TruncatedSVD(n_components=50)
X_reduced = svd.fit_transform(X)

kmeans = KMeans(n_clusters=5, random_state=42)
df["cluster"] = kmeans.fit_predict(X_reduced)

print("Clustering complete.")

NOTEEVENTS = pd.read_csv("NOTEEVENTS.csv")

def clean_text(text):
    if isinstance(text, str):
        text = re.sub(r"\[\*\*.*?\*\*\]", " ", text)
        text = re.sub(r"[^a-zA-Z\s]", " ", text)
        text = text.lower()
        text = text.replace("\n", " ")

        return text
    else:
        return ""
NOTEEVENTS["cleaned_text"] = NOTEEVENTS["TEXT"].apply(clean_text)
print(NOTEEVENTS[["TEXT", "cleaned_text"]].head())

noteevents = pd.read_csv("NOTEEVENTS.csv")
admissions = pd.read_csv("ADMISSIONS.csv")
def clean_text(text):
    if isinstance(text, str):
        text = re.sub(r"\[\*\*.*?\*\*\]", " ", text)
        text = re.sub(r"[^a-zA-Z\s]", " ", text)
        text = text.lower()
        return text
    return ""
noteevents["cleaned_text"] = noteevents["TEXT"].apply(clean_text)
noteevents = noteevents.merge(admissions[["SUBJECT_ID", "HADM_ID",
"HOSPITAL_EXPIRE_FLAG"]],
                              on=["SUBJECT_ID", "HADM_ID"], how="left")
noteevents = noteevents.dropna(subset=["cleaned_text", "HOSPITAL_EXPIRE_FLAG"])
X = noteevents["cleaned_text"]
y = noteevents["HOSPITAL_EXPIRE_FLAG"]
vectorizer = TfidfVectorizer(max_features=10000, stop_words='english')
X_tfidf = vectorizer.fit_transform(X)
X_train, X_test, y_train, y_test = train_test_split(X_tfidf, y, test_size=0.2, random_state=42)
model = LogisticRegression(max_iter=1000)
```

```
model.fit(X_train, y_train)
y_pred = model.predict(X_test)
print(classification_report(y_test, y_pred))

model = LogisticRegression(max_iter=1000, class_weight='balanced')

smote = SMOTE(random_state=42)
X_resampled, y_resampled = smote.fit_resample(X_tfidf, y)

ratio = (y == 0).sum() / (y == 1).sum()
from xgboost import XGBClassifier
import xgboost as xgb
model = xgb.XGBClassifier(scale_pos_weight=ratio, use_label_encoder=False,
eval_metric='logloss')

ratio = (y == 0).sum() / (y == 1).sum()
model = xgb.XGBClassifier(scale_pos_weight=ratio, use_label_encoder=False,
eval_metric='logloss')
model.fit(X_train, y_train)

print("Loading data...")
df = pd.read_csv("NOTEEVENTS.csv").dropna(subset=["TEXT"]).head(3000)
print(" Loading NLP model...")
nlp = spacy.load("en_core_web_sm")
def clean_note(text):
    doc = nlp(str(text))
    tokens = []
    for token in doc:
        word = token.lemma_.lower()
        if token.is_stop or token.is_punct or token.like_num:
            continue
        if re.match(r"^\d+[a-z]*$", word):
            continue
        tokens.append(word)
    return " ".join(tokens)
print("Preprocessing 3000 clinical notes...")
df["clean_text"] = df["TEXT"].apply(clean_note)

print("Vectorizing text...")
vectorizer = TfidfVectorizer(max_features=1000)
X = vectorizer.fit_transform(df["clean_text"])
print(" Reducing dimensions...")
svd = TruncatedSVD(n_components=100, random_state=42)
X_reduced = svd.fit_transform(X)
print(" Clustering into topics...")
kmeans = KMeans(n_clusters=5, random_state=42)
```

```
df["cluster"] = kmeans.fit_predict(X_reduced)

print("\n Top keywords per cluster (excluding numeric tokens):")
terms = vectorizer.get_feature_names_out()
order_centroids = kmeans.cluster_centers_.argsort()[:, ::-1]

for i in range(5):
    print(f"\nCluster {i + 1}:")
    keywords = []
    for ind in order_centroids[i]:
        word = terms[ind]
        if not re.match(r"^\d+[a-z]*$", word):  # filter numeric-like terms
            keywords.append(word)
        if len(keywords) == 10:
            break
    print(", ".join(keywords))
```

st.py (streamlit code)

```
import streamlit as st
import joblib
import numpy as np
import pandas as pd
rf_model = joblib.load("random_forest_model.pkl")
rf_label_encoders = joblib.load("rf_label_encoders.pkl")
los_model = joblib.load("los_model.pkl")
los_label_encoders = joblib.load("los_label_encoders.pkl")
@st.cache_data
def get_diagnosis_options():
    df = pd.read_csv("MERGED_DATA_cleaned_final1.csv")
    all_diagnoses = df["DIAGNOSIS"].dropna().astype(str).str.upper()

    simplified = all_diagnoses.str.split(r"[;,/]", expand=True).stack().str.strip()

    simplified = simplified[simplified.str.len() > 2]
    simplified = simplified[simplified.str.contains(r"[A-Z]{2,}", regex=True)]

    blacklist = [
        "P FALL", "P MOTOR VEHICLE ACCIDENT", "MOTOR VEHICLE ACCIDENT",
"FALL", "TRAUMA", "ASSAULT",
        "UNKNOWN", "MULTIPLE", "NONE", "UNSPECIFIED", "OTHER"
    ]
    simplified = simplified[~simplified.isin(blacklist)]

    top_diagnoses = simplified.value_counts().nlargest(20).index.tolist()
    top_diagnoses.append("Other")
```

```
    return sorted(top_diagnoses)

diagnosis_options = get_diagnosis_options()

st.title("🏥 Patient Admission & Length of Stay Prediction")
st.markdown("Enter patient information below:")

age = st.number_input("Age", min_value=0, max_value=120, value=50)
gender = st.selectbox("Gender", ["M", "F"])
heartrate = st.number_input("Heart Rate (bpm)", min_value=30, max_value=200, value=80)
resprate = st.number_input("Respiratory Rate (rpm)", min_value=5, max_value=60,
value=20)
o2sat = st.number_input("Oxygen Saturation (%)", min_value=50, max_value=100,
value=95)
temp = st.number_input("Temperature (°F)", min_value=85.0, max_value=110.0,
value=98.6)
sbp = st.number_input("Systolic BP (mmHg)", min_value=60, max_value=250, value=120)
dbp = st.number_input("Diastolic BP (mmHg)", min_value=30, max_value=150, value=80)
diagnosis = st.selectbox("Diagnosis", diagnosis_options)

def safe_encode(encoders, col, val):
    try:
        return encoders[col].transform([val])[0]
    except:
        return 0

input_dict = {
    "AGE": age,
    "GENDER": safe_encode(rf_label_encoders, "GENDER", gender),
    "HEARTRATE": heartrate,
    "RESPRATE": resprate,
    "O2SATURATION": o2sat,
    "TEMPERATURE": temp,
    "SYSTOLICBP": sbp,
    "DIASTOLICBP": dbp,
    "DIAGNOSIS": safe_encode(rf_label_encoders, "DIAGNOSIS", diagnosis)
}

if st.button("Predict Admission"):
    input_array = np.array([list(input_dict.values())])
    prediction = rf_model.predict(input_array)[0]
    proba = rf_model.predict_proba(input_array)[0][1]

    if prediction == 1:
        st.error(f" Patient is likely to be admitted (Probability: {proba:.2%})")
```

```
    # Prepare input for LOS prediction
    los_input = {
        "AGE": age,
        "GENDER": safe_encode(los_label_encoders, "GENDER", gender),
        "HEARTRATE": heartrate,
        "RESPRATE": resprate,
        "O2SATURATION": o2sat,
        "TEMPERATURE": temp,
        "SYSTOLICBP": sbp,
        "DIASTOLICBP": dbp,
        "DIAGNOSIS": safe_encode(los_label_encoders, "DIAGNOSIS", diagnosis)
    }

    los_array = np.array([list(los_input.values())])
    log_pred = los_model.predict(los_array)[0]
    los_days = np.expm1(log_pred)  # Reverse log1p transformation

    st.warning(f"Estimated Length of Stay: **{los_days:.1f} days**")

else:
    st.success(f"Patient is unlikely to be admitted (Probability: {proba:.2%})")
```