Of course. Here is the complete, final documentation for the Emergency Severity Prediction model we built together.

Project Documentation: Emergency Severity Prediction Model

1. Project Overview

- **Project Name**: EMsync: Emergency Coordination Hub [1]

- **Model Name**: Emergency Severity Prediction
- **Objective**: To predict the severity of an incoming emergency patient using their vital signs. The model classifies patients into one of three categories: **Critical**, **Moderate**, or **Low Urgency**. This allows hospital staff to prepare resources in advance, reducing treatment delays.
- **Primary Metric**: Due to the highly imbalanced nature of the dataset, the key performance metric is **Balanced Accuracy**. Our final model achieves a strong baseline score on this metric.

---

2. Data Source

- **Dataset**: MIMIC-IV-ED (Emergency Department) from PhysioNet [2][2][2][2]

- **Key Raw Files Used**:
  - edstays.csv
  - triage.csv
  - vitalsign.csv
- **Target Variable**: The acuity column from triage.csv was used as the ground truth. It was originally on a 5-point scale and was mapped to our 3 urgency levels.

---

3. Final Model Details

- **Model Type**: XGBoost Classifier
- **Final Performance**:
  - **Balanced Accuracy**: **71%**
  - **Macro F1-Score**: **54%**
- **Features Used**: The model was trained on the following 20 features:
  - temperature, heartrate, resprate, o2sat, sbp, dbp
  - heartrate_median, heartrate_min, heartrate_max
  - resprate_median
  - o2sat_median, o2sat_min
  - temperature_median
  - sbp_median, dbp_median
  - pain_median
  - shock_index, pulse_pressure (Engineered)
  - hypoxia_flag, fever_flag (Engineered)
- **Saved Model File**: xgboost_baseline_71bal_acc.pkl

---

4. Development Summary & Key Decisions

1. **Environment Setup**: An Anaconda environment (emsync_env) was created with Python 3.9 and all necessary libraries (pandas, scikit-learn, xgboost, etc.).
2. **Initial Debugging**: We resolved a KeyError by discovering the target variable in the source data was named acuity, not esi. The preprocessing scripts were updated accordingly.
3. **Handling Class Imbalance**: This was the most critical step. The model's initial performance was poor because the 5 ESI levels were severely imbalanced. We solved this by mapping the 5 levels into 3 broader urgency categories, which significantly improved the model's ability to learn.
4. **Model Selection**: We trained both Random Forest and XGBoost models. The XGBoost model was selected as the best performer because it achieved a much higher **Balanced Accuracy (71%)**, indicating it was more effective at identifying patients across all three categories, including the rare ones.
5. **Experimentation**: We conducted experiments with hyperparameter tuning and feature selection. We concluded that our initial baseline model with all 20 features was the most robust and reliable for this project's goals.
6. **Finalization**: The code was rolled back to the best-performing baseline version, and the final model was saved.

---

5. How to Use the Model (Inference)

The following script, predict_severity.py, demonstrates how to load the saved model and use it to predict the severity for a new patient based on their vitals.

Python

```
import joblib
import pandas as pd
import numpy as np

# --- 1. LOAD THE SAVED MODEL ---
model_filename = 'xgboost_baseline_71bal_acc.pkl'
print(f"Loading model: {model_filename}...")
model = joblib.load(model_filename)
print("✅ Model loaded successfully.")


# --- This is the same function from your training script ---
def add_engineered_features(df):
    df = df.copy()
    if {"heartrate", "sbp"}.issubset(df.columns):
        sbp_safe = df["sbp"].replace(0, np.nan)
        df["shock_index"] = df["heartrate"] / sbp_safe
    if {"sbp", "dbp"}.issubset(df.columns):
        df["pulse_pressure"] = df["sbp"] - df["dbp"]
    if "o2sat" in df.columns:
        df["hypoxia_flag"] = (df["o2sat"] < 90).astype(int)
    if "temperature" in df.columns:
        df["fever_flag"] = (df["temperature"] >= 38).astype(int)
    return df


# --- 2. PREPARE THE NEW PATIENT'S DATA ---
# Example vitals for one patient.
new_patient_data = {
    'temperature': 39.1, 'heartrate': 125, 'resprate': 28, 'o2sat': 88,
    'sbp': 95, 'dbp': 60, 'heartrate_median': 125, 'heartrate_min': 120,
    'heartrate_max': 130, 'resprate_median': 28, 'o2sat_median': 89,
```

```
    'o2sat_min': 87, 'temperature_median': 39.1, 'sbp_median': 95,
    'dbp_median': 60, 'pain_median': 8, 'shock_index': np.nan,
    'pulse_pressure': np.nan, 'hypoxia_flag': np.nan, 'fever_flag': np.nan
}

input_df = pd.DataFrame([new_patient_data])
input_df_engineered = add_engineered_features(input_df)

original_features = [
    'temperature', 'heartrate', 'resprate', 'o2sat', 'sbp', 'dbp',
    'heartrate_median', 'heartrate_min', 'heartrate_max', 'resprate_median',
    'o2sat_median', 'o2sat_min', 'temperature_median', 'sbp_median',
    'dbp_median', 'pain_median', 'shock_index', 'pulse_pressure',
    'hypoxia_flag', 'fever_flag'
]
input_df_final = input_df_engineered[original_features]


# --- 3. MAKE THE PREDICTION ---
prediction_index = model.predict(input_df_final)[0]
prediction_probabilities = model.predict_proba(input_df_final)[0]

severity_map = {0: 'Critical', 1: 'Moderate', 2: 'Low Urgency'}
predicted_severity = severity_map[prediction_index]


# --- 4. DISPLAY THE RESULT ---
print("\n--- PREDICTION RESULT ---")
print(f"Predicted Severity: {predicted_severity}")
print("Confidence Scores:")
print(f"  - Critical:    {prediction_probabilities[0]:.2%}")
print(f"  - Moderate:    {prediction_probabilities[1]:.2%}")
print(f"  - Low Urgency:  {prediction_probabilities[2]:.2%}")
print("-------------------------\n")
```