DM HW2 Predicting Patient Mortality (`has_died`) Report

---

1. Introduction

這次作業的主要目標是構建一個模型來預測住院病人的生存情況（`has_died`），其中 `0` 表示存活，`1` 表示死亡。

資料集結構：

- **訓練集：** 44,939 筆資料，包含患者的各種臨床和人口統計特徵，以及標籤 `has_died`。
- **測試集：** 19,260 筆資料，不包含類別標籤，用於最終的模型評估。

---

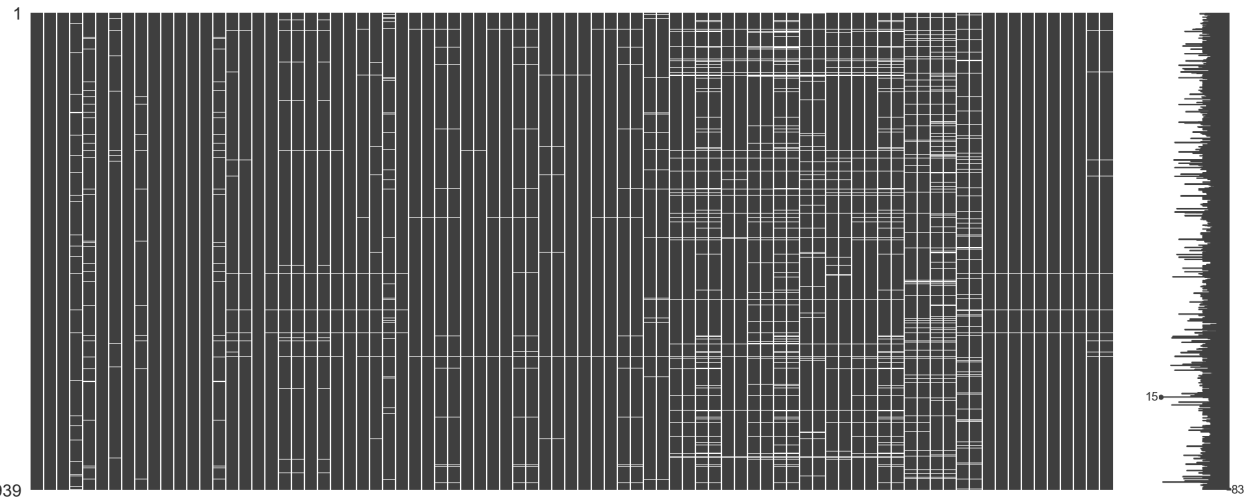
2. Data Preprocessing and Procedures

2.1 Dataset Analysis

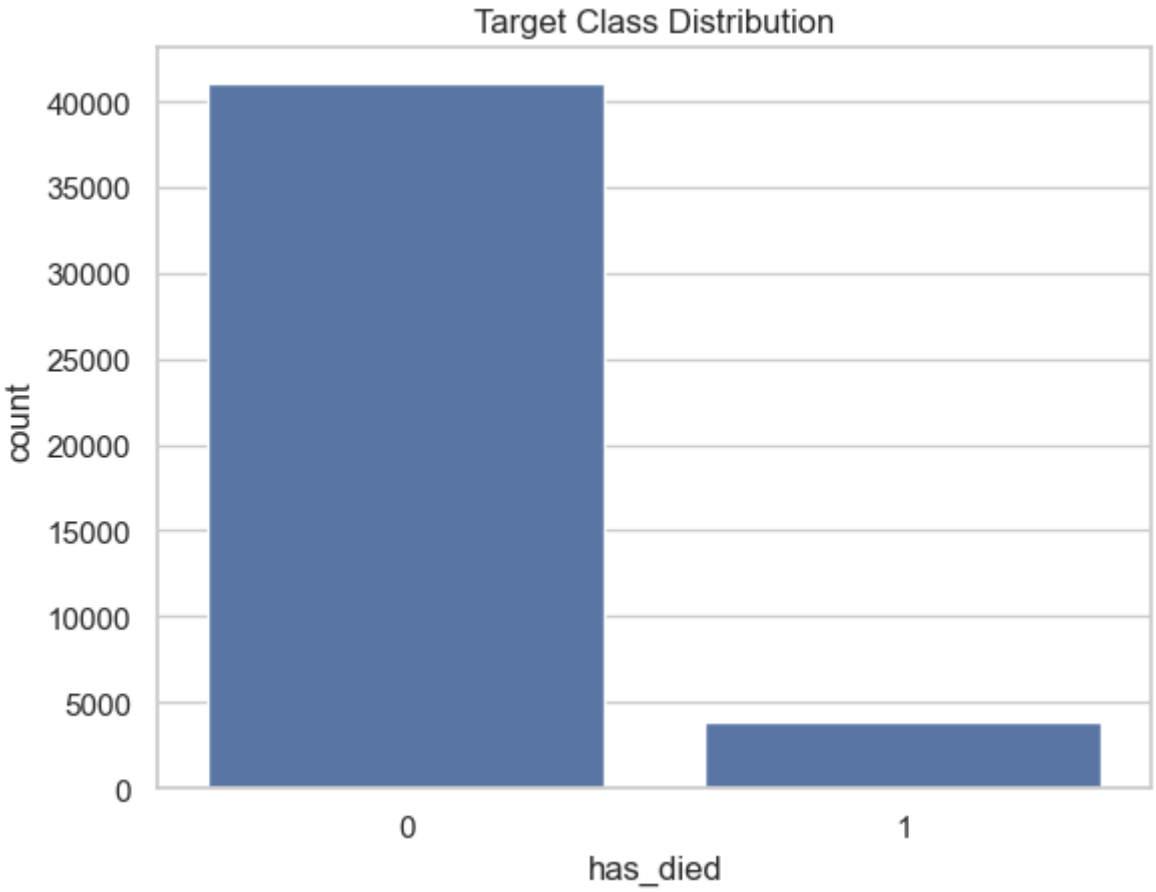
在分析此次的資料集後我有以下幾個發現:

1. 缺失值: 部分特徵如 `apache_4a_hospital_death_prob` 和 `apache_4a_icu_death_prob` 存在較高的缺失率，需要進行適當的插補(因為是特重要特徵選擇直接刪除效能會非常差)。

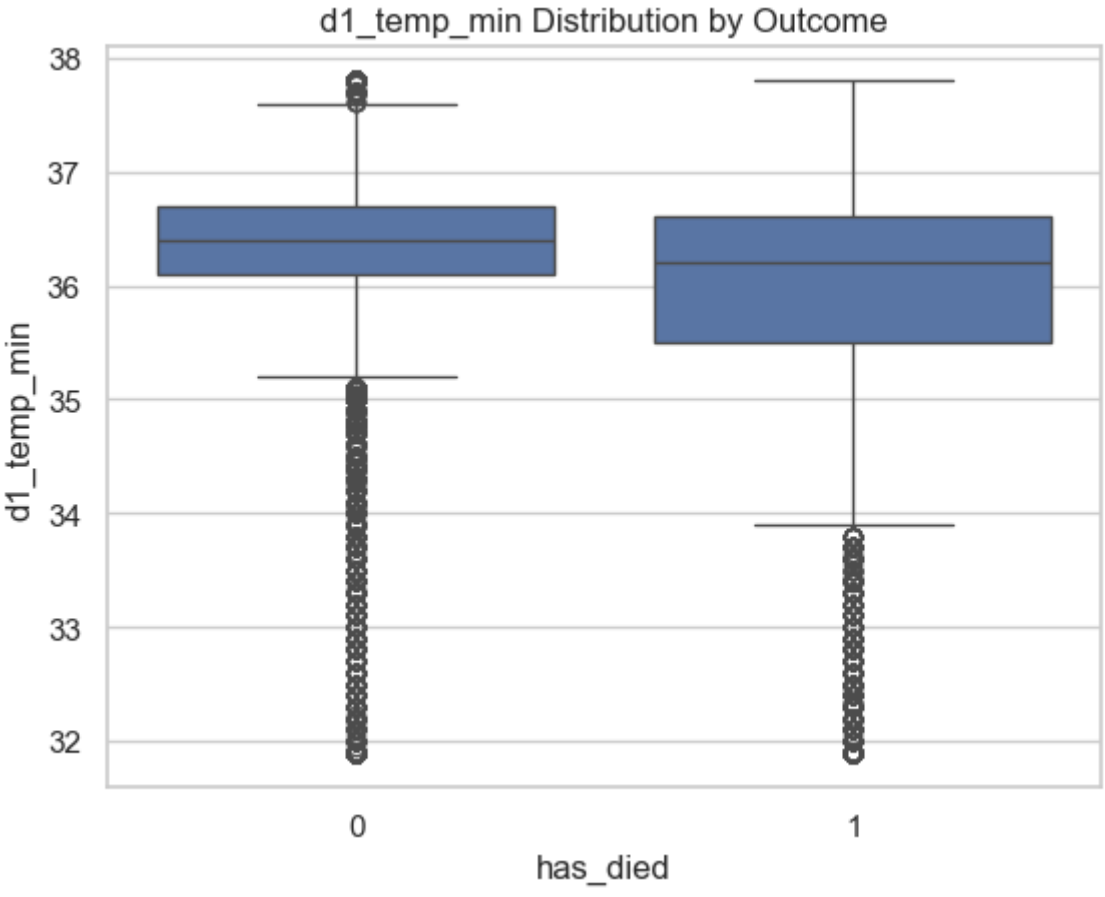
index	Missing Values	Percentage (%)
d1_potassium_min	4748.0	10.565433
d1_potassium_max	4748.0	10.565433
h1_mbp_noninvasive_min	4495.0	10.002448
h1_mbp_noninvasive_max	4495.0	10.002448
apache_4a_icu_death_prob	3891.0	8.658404



2. 資料不平衡: 訓練集中死亡樣本（`has_died = 1`）僅佔 20%，這表明模型需特別關注少數類別的表現，防止偏向多數類別，確保模型在識別高風險患者時的準確性和穩定性。



3. 異常值:`d1_temp_min` ≤ 33°C 的樣本佔比 1%，但死亡率高達 90%。這些樣本雖然數量少但在模型中可能產生極大影響，需要謹慎處理。



我將會基於這些發現去實現我在後續2.2和2.3的處理。

2.2 Data Cleaning

在進行模型訓練前，資料清理也是非常重要的，以下是我的清理相關操作以及緣由。

1. **移除不相關的列：**

   刪除 `encounter_id` 和 `patient_id`，因為這些資訊對預測無直接幫助，且可能有overfitting 風險。

```
columns_to_drop = ['encounter_id', 'patient_id']
train_data = train_data.drop(columns=columns_to_drop, errors='ignore')
test_data = test_data.drop(columns=columns_to_drop, errors='ignore')
```

2. **處理負值和無效值：**

   某些數值特徵如 `pre_icu_los_days` 出現負值，這在實際情境中無意義。我將這些負值設置為零，以保持資料邏輯一致性。

```
for col in ['pre_icu_los_days']:
    train_data[col] = train_data[col].apply(lambda x: max(x, 0))
    test_data[col] = test_data[col].apply(lambda x: max(x, 0))
```

   對於像 `apache_4a_hospital_death_prob` 和 `apache_4a_icu_death_prob` 這些特徵中含有 `-1` 作為占位符，表示缺失或無效資料，將其替換為 `NaN` 以便進行後續插補。

```
for col in ['apache_4a_hospital_death_prob', 'apache_4a_icu_death_prob']:
    train_data[col] = train_data[col].replace(-1, np.nan)
    test_data[col] = test_data[col].replace(-1, np.nan)
```

3. **合併稀有類別：**

   將類別特徵中頻率較低的類別合併為「其他」，以減少特徵的稀疏性，提高模型的泛化能力。

```
def merge_rare_categories(train_data, test_data, threshold=1):
    categorical_columns = train_data.select_dtypes(exclude=
[np.number]).columns.tolist()

    for col in categorical_columns:
        freq = train_data[col].value_counts(normalize=True) * 100
        rare_labels = freq[freq < threshold].index
        train_data[col] = train_data[col].replace(rare_labels, '其他')
        test_data[col] = test_data[col].replace(rare_labels, '其他')

    return train_data, test_data
```

```
train_data, test_data = merge_rare_categories(train_data, test_data)
```

在這些資料清理後，接下來我進行了一些資料轉換以及選一些重要特徵出來，去加強訓練的效能。

2.3 Data Transformation and Feature Engineering

為了提升模型的性能，我對資料進行了必要的轉換和特徵工程。

1. 對數轉換：

對數值分佈偏斜的特徵如 `pre_icu_los_days` 應用了 `log1p` 轉換，以穩定變異數並使分佈更接近正態。

```
numerical_columns = train_data.select_dtypes(include=[np.number]).columns.tolist()

for col in numerical_columns:
    if (train_data[col] > 0).all():
        train_data[col] = np.log1p(train_data[col])
        test_data[col] = np.log1p(test_data[col])
```

2. 重要特徵： 在了解資料集並透過幾次訓練的重要特徵分析以後得到兩個非常重要的特徵，就是 `apache_4a_hospital_death_prob` 和 `apache_4a_icu_death_prob` ，他們是基於 Apache IV 評分系統的死亡風險預測指標，直接反映患者的病情嚴重程度和預後狀況。這些指標對高風險患者的識別至關重要，所以以下是我對這兩個特徵進行的特別處理。

2.1特徵分級：

在了解資料集並透過幾次訓練的重要特徵分析以後得到兩個非常重要的特徵，就是 `apache_4a_hospital_death_prob` 和 `apache_4a_icu_death_prob` ，他們是基於 Apache IV 評分系統的死亡風險預測指標，直接反映患者的病情嚴重程度和預後狀況。這些指標對高風險患者的識別至關重要。 所以我決定將 `apache_4a_hospital_death_prob` 根據預定的閾值轉換為類別風險水平（如 `低風險`、`中風險`、`高風險`）。

```
risk_bins = [0, 0.3, 0.7, 1.0]
risk_labels = ['低風險', '中風險', '高風險']

for dataset in [train_data, test_data]:
    dataset['apache_4a_hospital_death_prob_risk'] = pd.cut(
        dataset['apache_4a_hospital_death_prob'],
        bins=risk_bins,
        labels=risk_labels,
        include_lowest=True
    )
```

2.2創建複合特徵：

結合 apache_4a_hospital_death_prob 和 apache_4a_icu_death_prob，按指定權重計算出 composite_risk，並創建如 age*composite_risk 和 bmi*composite_risk 的交互特徵，以捕捉特徵間的協同效應。

```
weight_hospital = 0.8
weight_icu = 0.2

for dataset in [train_data, test_data]:
    dataset['composite_risk'] = (
        dataset['apache_4a_hospital_death_prob'] * weight_hospital +
        dataset['apache_4a_icu_death_prob'] * weight_icu
    )

    if 'age' in dataset.columns and 'composite_risk' in dataset.columns:
        dataset['age*composite_risk'] = dataset['age'] * dataset['composite_risk']
    if 'bmi' in dataset.columns and 'composite_risk' in dataset.columns:
        dataset['bmi*composite_risk'] = dataset['bmi'] * dataset['composite_risk']
```

3. 處理缺失值：

移除高缺失率特徵： 移除缺失率超過 30% 的特徵，如某些臨床指標，避免插補後引入大量不確定性。

```
```python
missing_ratio = train_data.isnull().mean() * 100
columns_to_drop_missing = missing_ratio[missing_ratio > 30].index.tolist()
train_data = train_data.drop(columns=columns_to_drop_missing)
test_data = test_data.drop(columns=columns_to_drop_missing)
```
```

創建缺失指標： 對於缺失率超過 5% 的特徵，創建二元指標（如 _missing），讓模型知道該特徵是否缺失，這有助於模型捕捉缺失值本身所帶來的信息。

```
    for col in train_data.columns:
        if train_data[col].isnull().mean() * 100 > 5:
            train_data[f'{col}_missing'] = train_data[col].isnull().astype(int)
            test_data[f'{col}_missing'] = test_data[col].isnull().astype(int)
```

2.4 Data Imputation and Data Imbalance Handling

我對數值特徵使用了多重插補（IterativeImputer），根據其他特徵預測並填補缺失的數值值，這種方法比簡單插補更能保留資料的內在關係。對於類別特徵，使用了眾數填補（SimpleImputer），保留主要類別的信息。

```
from sklearn.experimental import enable_iterative_imputer
from sklearn.impute import SimpleImputer, IterativeImputer
```

```
# 數值型特徵使用多重插補
numerical_imputer = IterativeImputer(random_state=SEED)

# 類別型特徵使用眾數填補
categorical_imputer = SimpleImputer(strategy='most_frequent')
```

由於訓練集中正負類別分佈不均，`has_died = 1`樣本僅佔總數的 20%，為了減少類別不平衡對模型訓練的影響，我使用了 ADASYN（Adaptive Synthetic Sampling）進行過採樣。

```
from imblearn.over_sampling import ADASYN
from imblearn.pipeline import Pipeline as ImbPipeline

pipelines['XGBoost'] = ImbPipeline(steps=[
    ('preprocessor', preprocessor),
    ('adasyn', ADASYN(sampling_strategy='minority', random_state=SEED,
n_jobs=-1)),
    ('classifier', XGBClassifier(
        eval_metric='logloss',
        random_state=SEED,
        verbosity=0,
        scale_pos_weight=scale_pos_weight,
        n_jobs=-1,
        tree_method='gpu_hist',  # 使用 GPU
    ))
])
```

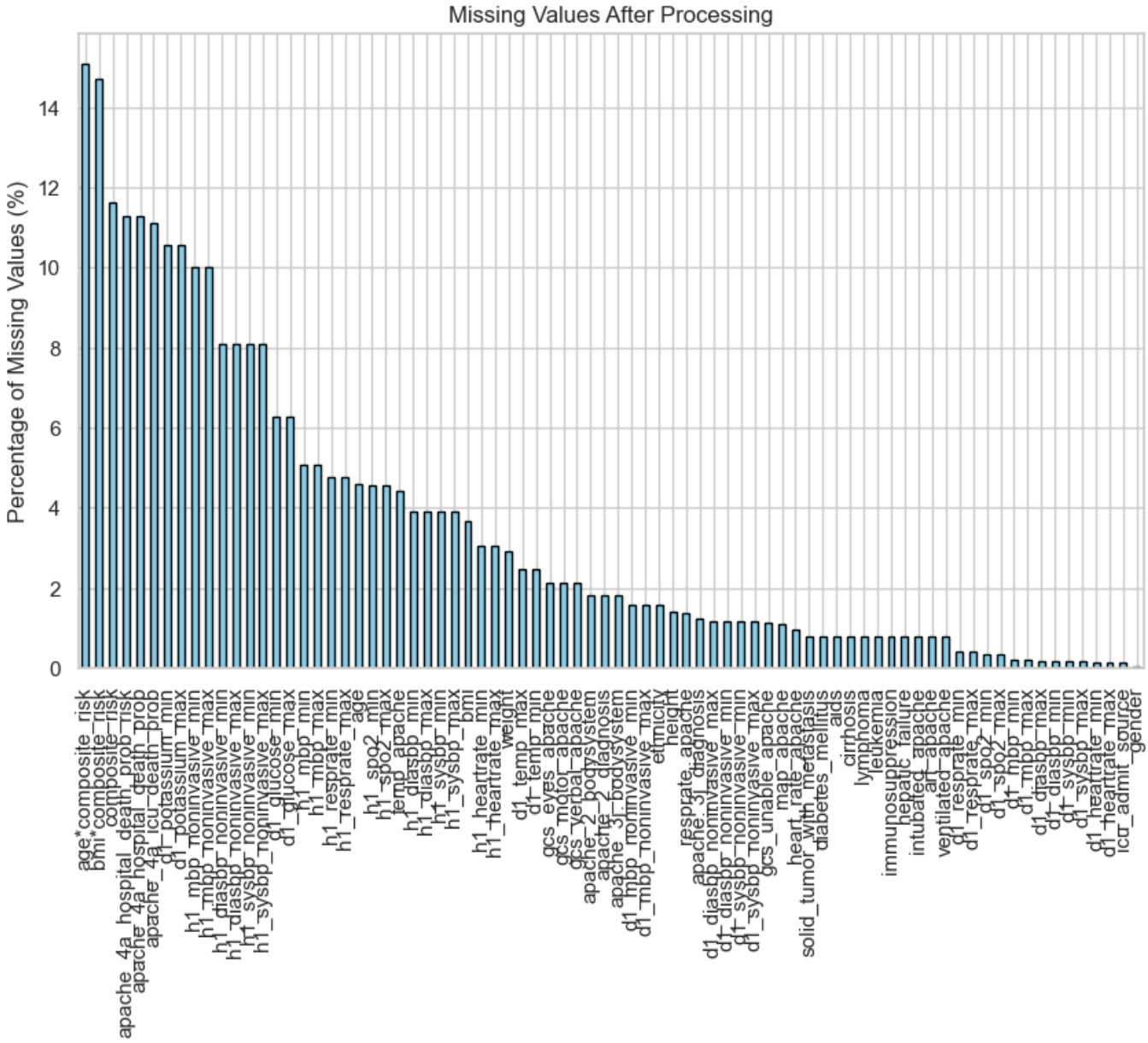
ADASYN 能夠生成少數類別的合成樣本，特別是在決策邊界附近生成，提升模型對複雜模式的學習能力。

2.5 Visualization Analysis

在綜合所有前處理後，我對我較為主要的前處理步驟繪製視覺化圖表，以方便查驗前處理後的資料狀況。

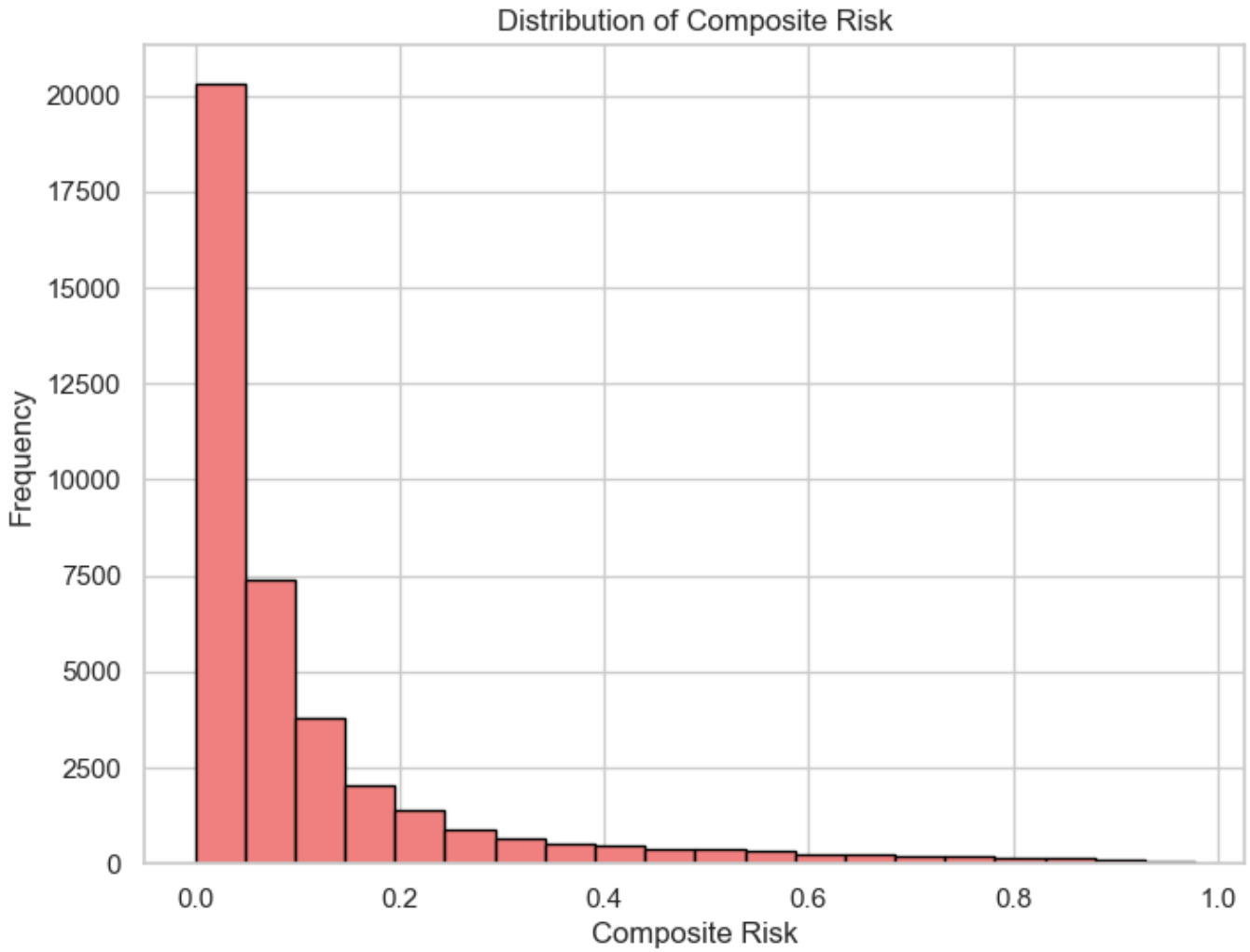
1. 缺失值處理後的結果

X 軸是特徵名稱，Y 軸是缺失值的百分比。

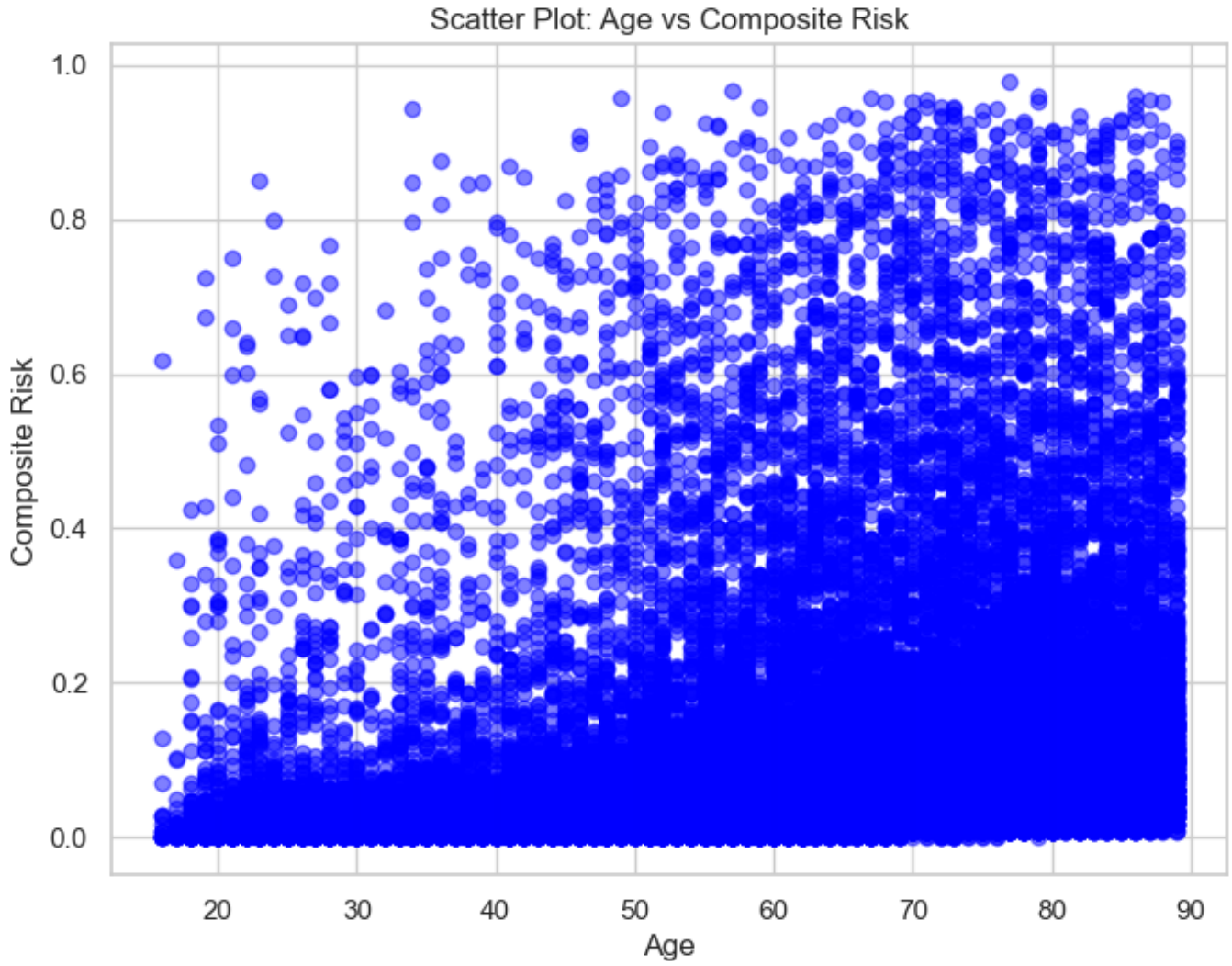


2. 衍生特徵的分佈

下圖是對`Composite Risk`的分佈直方圖，展示了該指標在整個患者群體中的頻率。大多數患者的綜合風險指標集中在較低的範圍內（接近 0.0），但仍存在少量風險較高的患者。



以下圖表顯示了患者的年齡（Age）與`Composite Risk`之間的分佈情況。每個點代表一名患者，隨著年齡增加，綜合風險指標呈現上升趨勢，特別是在年齡較大的群體中，風險分佈變得更加廣泛。



---

3. Classification Methods

3.1 Machine Learning Algorithms

XGBoost Classifier

- **描述：**

  XGBoost（Extreme Gradient Boosting）是一個優化的梯度提升框架，旨在提升性能和計算速度。它對結構化或表格資料特別有效，並內建處理缺失值和正則化功能以防止overfitting。XGBoost 通過加強學習過程中的誤差修正，能夠逐步提升模型的準確性。

- **reference：https://xgboost.readthedocs.io/en/latest/python/index.html**

- **可解釋性：**

  XGBoost 模型支持特徵重要性提取，並可與 SHAP（SHapley Additive exPlanations）結合，提供詳細的解釋性分析。這使得能夠理解每個特徵如何影響模型的預測，增強模型的透明度和可信度。

選擇 XGBoost 的原因：

- **高性能：**

在多數分類任務中表現優異，能夠處理大量資料和特徵，並且具有高準確度。

- **靈活性：**

  能夠高效處理數值型和類別型資料，並具備自動處理缺失值的能力，減少資料清理的工作量。

- **速度優勢：**

  針對速度和效能進行了優化，適用於大規模資料集，縮短訓練時間。

- **可解釋性：**

  與 SHAP 結合使用，可提供透明且易於理解的模型決策過程，便於醫療專業人員理解和信任模型的預測結果。

---

4. Results and Analysis

4.1 Model Performance

在完成資料預處理和特徵工程後用以下程式碼以8:2比例切分出internal validation set

```
  def split_data(train_data, train_target):
      """分割資料集並返回訓練集與驗證集"""
      X = train_data
      y = train_target['has_died']

      # 分割訓練集和驗證集
      X_train, X_val, y_train, y_val = train_test_split(
          X, y, test_size=0.2, random_state=SEED, stratify=y
      )

      logging.info(f"Data split into training and validation sets. Training shape:
{X_train.shape}, Validation shape: {X_val.shape}")

      return X_train, X_val, y_train, y_val

  X_train, X_val, y_train, y_val = split_data(train_data, train_target)
```

並使用 XGBoost 分類器進行模型訓練。通過 5 折交叉驗證評估模型的性能，主要關注macro F1-Score 和 AUROC 指標。

```
def hyperparameter_tuning(pipelines, X_train, y_train):
    # ... 省略部分代碼
    for name, pipeline in pipelines.items():
        # ... 省略部分代碼
        scores = cross_val_score(pipeline, X_train, y_train, cv=5,
scoring='f1_macro', n_jobs=-1)
        return scores.mean()
```

- **平均macro F1-Score：** 0.72
- **平均 AUROC（ROC 曲線下面積）：** 0.8 **交叉驗證結果：**

```
Trial 3 finished with value: 0.7212037538610677 and parameters:
{'classifier__n_estimators': 482, 'classifier__max_depth': 10,
'classifier__learning_rate': 0.037431371850834984, 'classifier__subsample':
0.8396809029533532, 'classifier__colsample_bytree': 0.7037348841325521}. Best is
trial 3 with value: 0.7212037538610677.
```

驗證後效能報告：

```
XGBoost - Validation F1 Score: 0.728371093557389
XGBoost - Validation AUROC: 0.8873725149517177
XGBoost - Validation Precision: 0.5246132208157525
XGBoost - Validation Recall: 0.4806701030927835
XGBoost - Validation PR AUC: 0.5117834975008115

Classification Report:
              precision    recall  f1-score   support

           0       0.95      0.96      0.96      8212
           1       0.52      0.48      0.50       776

    accuracy                           0.92      8988
   macro avg       0.74      0.72      0.73      8988
weighted avg       0.91      0.92      0.92      8988


Confusion Matrix:
 [[7874  338]
 [ 403  373]]
```

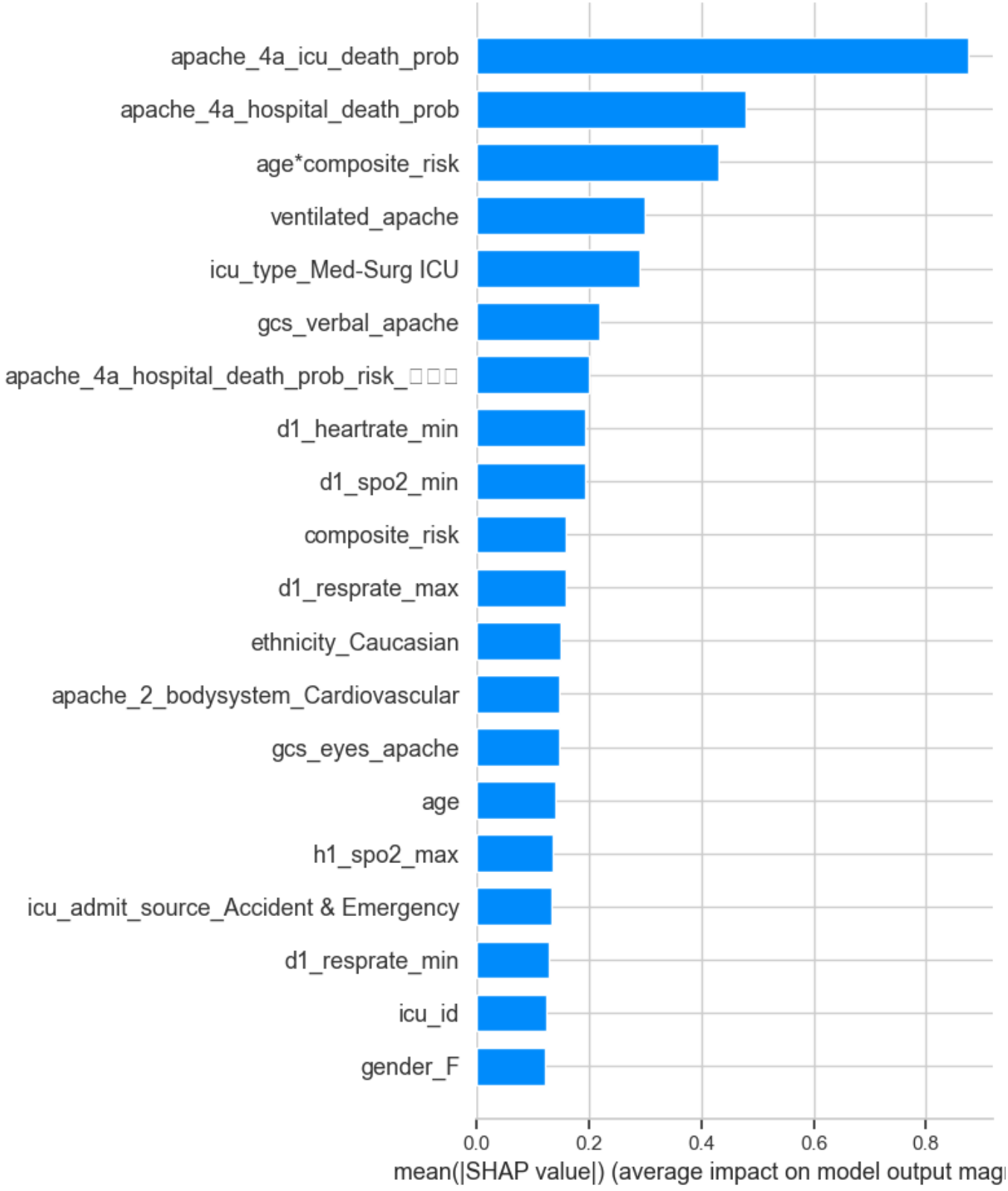
- **混淆矩陣分析：**

  在驗證集中，模型成功識別了 72 % 的正例（`has_died = 1`），但仍有 18% 的正例被誤分類為負例（`has_died = 0`）。

4.2 Explainable Experiment

為了更好地理解模型的決策過程，使用 SHAP（SHapley Additive exPlanations）進行特徵重要性分析，識別出對模型預測影響最大的前 20 個特徵。

Top 20 最重要的特徵：



相關特徵解釋；

- **複合風險指標（`composite_risk`）：**

  作為最具影響力的特徵，`composite_risk` 綜合了患者在醫院和 ICU 的死亡概率，顯示了其在預測死亡率中的關鍵作用。

- **生理指標：**

如 `d1_sysbp_noninvasive_min`（收縮壓最低值）和 `d1_temp_min`（體溫最低值）等臨床指標對預測有顯著影響，反映了這些生理參數在患者健康狀況中的重要性。

- **特徵交互的臨床：**
  - **年齡與綜合風險交互：**

    年齡和 `composite_risk` 的交互作用表明，年長且具有高綜合風險分數的患者風險呈指數級增長。

關鍵觀察：

- **正向影響死亡率的特徵：**
  - 較高的 `composite_risk`、年齡、異常的 BMI、低收縮壓和低體溫顯著增加死亡的可能性。
- **負向影響死亡率的特徵：**
  - 某些婚姻狀況類別（如已婚）可能與較低的死亡風險相關，這可能是因為已婚患者擁有更多的社會支持。

4.3 遇到的問題和挑戰

在調校模型的過程中我不斷嘗試新的方法去提高效能，較為值得一提的是，在調整超參數以外，我曾嘗試過使用ensemble方法，先將一個模型針對最重要的特徵`apache_4a_hospital_death_prob` 和 `apache_4a_icu_death_prob`按指定權重計算出的 `composite_risk`作為唯一特徵帶入第一個模型訓練，並使用邏輯回歸模型作為訓練模型，並且這個模型可以獲得F1-SCORE約為0.66的效能，我希望將這個模型和另一個模型做ensemble(staking)訓練，選擇staking的原因是這兩個模型其實本質上並沒有太多相輔相成的效果(即視作相同類型的模型)，由於只有兩個模型且性質相似，再做voting的意義不大，所以選擇staking作為我的ensemble方法。但是由於本來的模型進行參數調整時再配對上一號模型會造成參數極多的狀況，並且還需要再另外訓練一個模型(ensemble)，在考量硬體能力及配分狀況還有個人的時間分配後，決定放棄此方法，

5. Reproducing the Results

通過對資料的深入分析和預處理，以及使用 XGBoost 模型進行訓練和優化，我成功地構建了一個能夠高效且可解釋地預測患者死亡率的模型。SHAP 分析幫助我理解了特徵對模型預測的影響，增強了模型的透明度和可信度。

相關套件：

- **資料處理：**
  - `pandas`, `numpy`
- **可視化：**
  - `matplotlib`, `seaborn`, `shap`, `missingno`
- **機器學習：**
  - **預處理與建模：**
    - `scikit-learn`, `imbalanced-learn`

    - **提升算法：**
        - `xgboost`
    - **超參數調整：**
        - `optuna`
- **模型保存：**
    - `joblib`
- **日誌記錄：**
    - `logging`
- **配置管理：**
    - `yaml`

1. **環境設置：**
    - 使用 `conda` 或 `pip` 安裝必要的套件。所需套件在提供的腳本開頭列出。

    ```
    conda install pandas matplotlib seaborn scikit-learn imbalanced-learn
    pyyaml shap xgboost joblib
    ```

2. **專案結構：**
    - 確保以下目錄結構：

    ```
    project/
    ├── config/
    │   └── config.yaml
    ├── data/
    │   ├── train_X.csv
    │   ├── train_y.csv
    │   └── test_X.csv
    ├── logs/
    ├── trained_best_model.pkl
    ├── submission.csv
    └── model_api.py
    ```

3. **配置文件：**
    - 在 `config/config.yaml` 中填入資料集的正確路徑。例如：

    ```
    data_paths:
      train_X: "data/train_X.csv"
    ```

```
    train_y: "data/train_y.csv"
    test_X: "data/test_X.csv"
```

4. **運行腳本：**
    - 在 Jupyter Notebook 或 Python 環境中執行提供的腳本。
5. **模型部署：**
    - 執行 `313554059.py` 腳本以啟動 Flask 服務器進行模型推論：

```
python 313554059.py
```

6. **日誌查看：**
    - 查看 `logs/training_log.log` 以獲取訓練和評估過程的詳細日誌。
