Introduction

This study focuses on predicting 30-days mortality for MIMIC-III patients diagnosed with sepsis-3 using the XGBoost algorithm. This is a predictive modeling problem aiming to improve mortality prediction over traditional methods. The importance of solving this problem is aimed at tackling sepsis, which is a major cause of mortality especially in ICU patients. Early and accurate prediction of mortality in these patients is important as it can guide time-critical and appropriate treatment for patients, potentially improving survival outcomes. This study aims to enhance the predictive accuracy using machine learning, which could provide clinicians with a powerful tool for risk assessment and management.

The complexity of sepsis, including its vague syndrome definitions, unknown sources of infection, as well as high mortality rates, makes establishing a reliable and effective prognostic model challenging. Traditional machine learning models, which are based on small sample sizes or simplistic statistical assumptions, are limited in their predictive power. Traditional methods for diagnosing sepsis include the use of serum markers and scoring systems like APHACHE-II and SAPS-II. However, these have limitations in sensitivity, specificity, and the ability to handle complex interactions within data. XGBoost has shown improved predictive performance by efficiently handling missing data and assembling weak prediction models to create a more accurate composite model.

L. et al. [1] proposed a machine learning model using the XGBoost algorithm to predict 30-day mortality among patients with sepsis-3 in the MIMIC-III dataset, comparing its performance against traditional logistic regression and SAPS-II score models. The innovation for this study is found in applying the XGBoost algorithm, known for its efficiency with missing data and capability to enhance predictive accuracy by combining weak models. This study demonstrates the superiority of XGBoost over conventional methods in the context of sepsis mortality prediction.

For post-results analysis and scoring, tthe XGBoost model showed superior performance with higher AUCs compared to traditional logistic regression and SAPS-II models, indicating better predictive accuracy. This study significantly contributes to the sepsis research field by showcasing the application of a machine learning algorithm to accurately predict mortality, potentially aiding clinicians in making informed decisions and tailoring patient management strategies effectively.

Scope of Reproducibility:

1. Hypothesis 1: The XGBoost algorithm can outperform traditional logistic regression and SAPS-II scoring models in predicting 30-day mortality among patients with sepsis-3 based on scoring metrics.

- Corresponding experiment: Create a logistic regression model, a SAPS-II dataset, and an XGBoost model, and use the features specified by L. et al. [1] for these models. Then, analyze the results of both models in predicting 30-day mortality due to sepsis-3 and compare.

2. Hypothesis 2: The set of features identified by the XGBoost model as significant predictors of 30-day mortality are significant compared to features not used

- Corresponding experiment: Perform several ablations, including dropping several features and adding several others, and compare post-training scores across different models to see which features area most important in each model's decisions, and how effective each model is.

Methodology

This methodology is the core of your project. It consists of run-able codes with necessary annotations to show the expeiment you executed for testing the hypotheses.

The methodology at least contains two subsections **data** and **model** in your experiment.

```
!pip install google-cloud-bigquery
!pip install pandas
!pip install db-dtypes
!pip install scikit-learn
!pip install xgboost
!pip install opencv-python
!pip install matplotlib
```

```
Requirement already satisfied: google-cloud-bigquery in /usr/local/lib/python3.10/dist-packages (3.12.0)
Requirement already satisfied: grpcio<2.0dev,>=1.47.0 in /usr/local/lib/python3.10/dist-packages (from google-cloud-bigquery) (1.62.1)
Requirement already satisfied: google-api-core[grpc]!=2.0.*,!=2.1.*,!=2.2.*,!=2.3.0,<3.0.0dev,>=1.31.5 in /usr/local/lib/python3.10/dist
Requirement already satisfied: proto-plus<2.0.0dev,>=1.15.0 in /usr/local/lib/python3.10/dist-packages (from google-cloud-bigquery) (1.2
Requirement already satisfied: google-cloud-core<3.0.0dev,>=1.6.0 in /usr/local/lib/python3.10/dist-packages (from google-cloud-bigquery
Requirement already satisfied: google-resumable-media<3.0dev,>=0.6.0 in /usr/local/lib/python3.10/dist-packages (from google-cloud-bigqu
```

```
Requirement already satisfied: packaging>=20.0.0 in /usr/local/lib/python3.10/dist-packages (from google-cloud-bigquery) (24.0)
Requirement already satisfied: protobuf!=3.20.0,!=3.20.1,!=4.21.0,!=4.21.1,!=4.21.2,!=4.21.3,!=4.21.4,!=4.21.5,<5.0.0dev,>=3.19.5 in /us
Requirement already satisfied: python-dateutil<3.0dev,>=2.7.2 in /usr/local/lib/python3.10/dist-packages (from google-cloud-bigquery) (2
Requirement already satisfied: requests<3.0.0dev,>=2.21.0 in /usr/local/lib/python3.10/dist-packages (from google-cloud-bigquery) (2.31.
Requirement already satisfied: googleapis-common-protos<2.0.dev0,>=1.56.2 in /usr/local/lib/python3.10/dist-packages (from google-api-co
Requirement already satisfied: google-auth<3.0.dev0,>=2.14.1 in /usr/local/lib/python3.10/dist-packages (from google-api-core[grpc]!=2.0
Requirement already satisfied: grpcio-status<2.0.dev0,>=1.33.2 in /usr/local/lib/python3.10/dist-packages (from google-api-core[grpc]!=2
Requirement already satisfied: google-crc32c<2.0dev,>=1.0 in /usr/local/lib/python3.10/dist-packages (from google-resumable-media<3.0dev
Requirement already satisfied: six>=1.5 in /usr/local/lib/python3.10/dist-packages (from python-dateutil<3.0dev,>=2.7.2->google-cloud-bi
Requirement already satisfied: charset-normalizer<4,>=2 in /usr/local/lib/python3.10/dist-packages (from requests<3.0.0dev,>=2.21.0->goo
Requirement already satisfied: idna<4,>=2.5 in /usr/local/lib/python3.10/dist-packages (from requests<3.0.0dev,>=2.21.0->google-cloud-bi
Requirement already satisfied: urllib3<3,>=1.21.1 in /usr/local/lib/python3.10/dist-packages (from requests<3.0.0dev,>=2.21.0->google-cl
Requirement already satisfied: certifi>=2017.4.17 in /usr/local/lib/python3.10/dist-packages (from requests<3.0.0dev,>=2.21.0->google-cl
Requirement already satisfied: cachetools<6.0,>=2.0.0 in /usr/local/lib/python3.10/dist-packages (from google-auth<3.0.dev0,>=2.14.1->go
Requirement already satisfied: pyasn1-modules>=0.2.1 in /usr/local/lib/python3.10/dist-packages (from google-auth<3.0.dev0,>=2.14.1->goo
Requirement already satisfied: rsa<5,>=3.1.4 in /usr/local/lib/python3.10/dist-packages (from google-auth<3.0.dev0,>=2.14.1->google-api-
Requirement already satisfied: pyasn1<0.7.0,>=0.4.6 in /usr/local/lib/python3.10/dist-packages (from pyasn1-modules>=0.2.1->google-auth<
Requirement already satisfied: pandas in /usr/local/lib/python3.10/dist-packages (2.0.3)
Requirement already satisfied: python-dateutil>=2.8.2 in /usr/local/lib/python3.10/dist-packages (from pandas) (2.8.2)
Requirement already satisfied: pytz>=2020.1 in /usr/local/lib/python3.10/dist-packages (from pandas) (2023.4)
Requirement already satisfied: tzdata>=2022.1 in /usr/local/lib/python3.10/dist-packages (from pandas) (2024.1)
Requirement already satisfied: numpy>=1.21.0 in /usr/local/lib/python3.10/dist-packages (from pandas) (1.25.2)
Requirement already satisfied: six>=1.5 in /usr/local/lib/python3.10/dist-packages (from python-dateutil>=2.8.2->pandas) (1.16.0)
Requirement already satisfied: db-dtypes in /usr/local/lib/python3.10/dist-packages (1.2.0)
Requirement already satisfied: packaging>=17.0 in /usr/local/lib/python3.10/dist-packages (from db-dtypes) (24.0)
Requirement already satisfied: pandas>=0.24.2 in /usr/local/lib/python3.10/dist-packages (from db-dtypes) (2.0.3)
Requirement already satisfied: pyarrow>=3.0.0 in /usr/local/lib/python3.10/dist-packages (from db-dtypes) (14.0.2)
Requirement already satisfied: numpy>=1.16.6 in /usr/local/lib/python3.10/dist-packages (from db-dtypes) (1.25.2)
Requirement already satisfied: python-dateutil>=2.8.2 in /usr/local/lib/python3.10/dist-packages (from pandas>=0.24.2->db-dtypes) (2.8.2
Requirement already satisfied: pytz>=2020.1 in /usr/local/lib/python3.10/dist-packages (from pandas>=0.24.2->db-dtypes) (2023.4)
Requirement already satisfied: tzdata>=2022.1 in /usr/local/lib/python3.10/dist-packages (from pandas>=0.24.2->db-dtypes) (2024.1)
Requirement already satisfied: six>=1.5 in /usr/local/lib/python3.10/dist-packages (from python-dateutil>=2.8.2->pandas>=0.24.2->db-dtyp
Requirement already satisfied: scikit-learn in /usr/local/lib/python3.10/dist-packages (1.2.2)
Requirement already satisfied: numpy>=1.17.3 in /usr/local/lib/python3.10/dist-packages (from scikit-learn) (1.25.2)
Requirement already satisfied: scipy>=1.3.2 in /usr/local/lib/python3.10/dist-packages (from scikit-learn) (1.11.4)
Requirement already satisfied: joblib>=1.1.1 in /usr/local/lib/python3.10/dist-packages (from scikit-learn) (1.4.0)
Requirement already satisfied: threadpoolctl>=2.0.0 in /usr/local/lib/python3.10/dist-packages (from scikit-learn) (3.4.0)
Requirement already satisfied: xgboost in /usr/local/lib/python3.10/dist-packages (2.0.3)
Requirement already satisfied: numpy in /usr/local/lib/python3.10/dist-packages (from xgboost) (1.25.2)
Requirement already satisfied: scipy in /usr/local/lib/python3.10/dist-packages (from xgboost) (1.11.4)
Requirement already satisfied: opencv-python in /usr/local/lib/python3.10/dist-packages (4.8.0.76)
Requirement already satisfied: numpy>=1.21.2 in /usr/local/lib/python3.10/dist-packages (from opencv-python) (1.25.2)
```

```
# Need to check if google colab or not to properly show images
import sys
GOOGLE_COLAB ='google.colab' in sys.modules
print('Using Google Colab: ', GOOGLE_COLAB)

if GOOGLE_COLAB:
  from google.colab.patches import cv2_imshow
```

```
Using Google Colab:  True
```

```
import io
import os
import sys
import cv2
import json
import requests
import zipfile

import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
from google.cloud import bigquery

# Set env variables
os.environ["GCLOUD_PROJECT"] = 'dlh-project-418923'
os.environ["BQ_DATASET_BASE"] = 'mimiciii_clinical'
os.environ["BQ_DATASET_DERIVED"] = 'mimiciii_derived'
os.environ["BQ_DATASET_FEATURES"] = 'models_features'
os.environ["LOG_REG_FEATURES_VIEW"] = 'logreg_features'
os.environ["XGBOOST_FEATURES_VIEW"] = 'xgboost_features'
os.environ["SAPSII_FEATURES_VIEW"] = 'sapsii_features'
RANDOM_STATE = 42

# URLs for downloading from drive
gcloud_ids = {'sa-creds': '1qNBEkuIQgj0K-PJ9ETvqF6Cel8wMlVGA',
              'mimic-views': '13uCK5Go9XvANkeujrDaWY2cK3ocWakVM',
              'model-features': '1_z6HLDKnHH8iUddS_wMPg3UhexhczpZq',
              'img-features': '1VTAdcxG0Tz6vLhRWhc2b_kWaHjfeK_am',
              'graph-roc': '1Hc_vLhGYR4yU4LG-o42p3rVdUEDHXAuR',
              'graph-dca': '1q6c4b_IWql7hp829K1dDaoS6GEP6Bhcv',
              'graph-cic': '1x56v9kFDM_dS3JcML3UL0BlOSoJr7XGx',
              'graph-nomogram': '1yyFhCauWCAgqRyJe5E-DZdkdfiJiOiIC',
              'my-roc': '1LHFSwLI1X5fOCVtPClQr9od6c3ZfD4GD',
              'my-dca': '1mVIHMui69o3SRv76cbrrm9CnnbsT7zWn',}
gcloud_url = 'https://drive.google.com/uc?export=download&id={}'


# Define a function to show images based on URL
def show_image(gcloud_id: str, resize: bool=False) -> None:
  # Download bytes from URL
  f = io.BytesIO()
  # Write to BytesIO and reset stream position
  f.write(requests.get(gcloud_url.format(gcloud_ids[gcloud_id])).content)
  f.seek(0)
  # Write bytes into numpy array, then to a cv2 image
  file_bytes = np.asarray(bytearray(f.read()), dtype=np.uint8)
  img = cv2.imdecode(file_bytes, cv2.IMREAD_COLOR)
  # Resize if specified
  if resize: img=cv2.resize(img, (800,600))
  # Close bytesIO
  f.close()

  # Show image
  if GOOGLE_COLAB: cv2_imshow(img)
  else: cv2.imshow(img)
```

Data

Data includes raw data (MIMIC III tables), descriptive statistics (our homework questions), and data processing (feature engineering).

- Source of the data: where the data is collected from; if data is synthetic or self-generated, explain how. If possible, please provide a link to the raw datasets.
- Statistics: include basic descriptive statistics of the dataset like size, cross validation split, label distribution, etc.
- Data process: how do you munipulate the data, e.g., change the class labels, split the dataset to train/valid/test, refining the dataset.
- Illustration: printing results, plotting figures for illustration.
- You can upload your raw dataset to Google Drive and mount this Colab to the same directory. If your raw dataset is too large, you can upload the processed dataset and have a code to load the processed dataset.

Source

The data used in this project is the MIMIC III dataset. However, due to the size of the original dataset, a subsample of roughly half of the raw original dataset was loaded into Google Bigquery. The reason BQ was chosen was due to the availability of this GitHub repo from MIT that has easily creatable views and analyses of the MIMIC datasets, and can be processed via BQ.

The data is split into 3 datasets in BQ:

1. **mimiciii_clinical**: All of the original, raw data. Data was sourced from Physionet
2. **mimiciii_derived**: Any additional views created from the raw data. SQL for these views was sourced from the mimic-code library mentioned above
3. **mimiciii_features**: Views created that contain the features for each type of model. These features were chosen from L. et al. [1][p.8] based on the author's chose features, which were, "identifed by the results of backward stepwise analysis and strongly associated with mortality in 30 days".

Statistics

Tables and views

Tables contained in 'dlh-project-418923.mimiciii_clinical':

- dlh-project-418923.mimiciii_clinical.admissions, 58976 rows
- dlh-project-418923.mimiciii_clinical.callout, 34499 rows
- dlh-project-418923.mimiciii_clinical.caregivers, 7567 rows
- dlh-project-418923.mimiciii_clinical.chartevents, 330712483 rows
- dlh-project-418923.mimiciii_clinical.cptevents, 573146 rows
- dlh-project-418923.mimiciii_clinical.d_cpt, 134 rows
- dlh-project-418923.mimiciii_clinical.d_icd_diagnoses, 14567 rows
- dlh-project-418923.mimiciii_clinical.d_icd_procedures, 3882 rows
- dlh-project-418923.mimiciii_clinical.d_items, 12487 rows
- dlh-project-418923.mimiciii_clinical.d_labitems, 753 rows
- dlh-project-418923.mimiciii_clinical.datetimeevents, 4285647 rows
- dlh-project-418923.mimiciii_clinical.diagnoses_icd, 651047 rows
- dlh-project-418923.mimiciii_clinical.drgcodes, 115557 rows
- dlh-project-418923.mimiciii_clinical.icustays, 61532 rows
- dlh-project-418923.mimiciii_clinical.inputevents_cv, 14527935 rows
- dlh-project-418923.mimiciii_clinical.inputevents_mv, 3218991 rows
- dlh-project-418923.mimiciii_clinical.labevents, 23851932 rows
- dlh-project-418923.mimiciii_clinical.microbiologyevents, 631726 rows
- dlh-project-418923.mimiciii_clinical.noteevents, 2083180 rows
- dlh-project-418923.mimiciii_clinical.outputevents, 4349218 rows
- dlh-project-418923.mimiciii_clinical.patients, 46520 rows
- dlh-project-418923.mimiciii_clinical.prescriptions, 4156450 rows
- dlh-project-418923.mimiciii_clinical.procedureevents_mv, 258066 rows
- dlh-project-418923.mimiciii_clinical.procedures_icd, 231945 rows
- dlh-project-418923.mimiciii_clinical.services, 73343 rows
- dlh-project-418923.mimiciii_clinical.transfers, 261897 rows

Views contained in 'dlh-project-418923.mimiciii_derived', as well as descriptions:

- dlh-project-418923.mimiciii_derived.blood_gas_first_day -- Highest and lowest blood gas values in the first 24 hours of a patient's ICU stay.
- dlh-project-418923.mimiciii_derived.blood_gas_first_day_arterial -- As above, but arterial blood gases only.
- dlh-project-418923.mimiciii_derived.echo_data -- Text extracted from echocardiography reports using regular expressions.
- dlh-project-418923.mimiciii_derived.elixhauser_ahrq_v37 -- Comorbidities in categories proposed by Elixhauser et al. AHRQ produced the mapping.
- dlh-project-418923.mimiciii_derived.explicit_sepsis -- Explicitly coded sepsis (i.e. a list of patients with ICD-9 codes which refer to sepsis).
- dlh-project-418923.mimiciii_derived.gcs_first_day -- Highest and lowest Glasgow Coma Scale in the first 24 hours of a patient's ICU stay.
- dlh-project-418923.mimiciii_derived.labs_first_day -- Highest and lowest laboratory values in the first 24 hours of a patient's ICU stay.
- dlh-project-418923.mimiciii_derived.sofa -- The Sequential Organ Failure Assessment (SOFA) scale.
- dlh-project-418923.mimiciii_derived.urine_output_first_day -- Total urine output over the first 24 hours of a patient's ICU stay.
- dlh-project-418923.mimiciii_derived.ventilation_classifications -- Classifies patient settings as implying mechanical ventilation.

- dlh-project-418923.mimiciii_derived.ventilation_durations -- Start and stop times for mechanical ventilation.
- dlh-project-418923.mimiciii_derived.vitals_first_day -- Highest and lowest vital signs in the first 24 hours of a patient's ICU stay.
- dlh-project-418923.mimiciii_derived.sapsii -- Simplified Acute Physiology Score II (SAPS II)

Views contained in 'dlh-project-418923.models_features', which are used for the 2 models in this analysis (logistic regression and XGBoost):

- dlh-project-418923.models_features.logreg_features
- dlh-project-418923.models_features.xgboost_features
- dlh-project-418923.models_features.sapsii_features

Features selected

From L. et al. [1][p.4] regarding feature selection:

"Firstly, the conventional logistic regression model was conducted using these signifcant variables identifed by backward stepwise analysis with Chi-square test. Then we chose an entry probability of p<0.05 by the stepwise selection method. Secondly, in the construction of SAPS II model, we used these time-stamp variables to do prediction based on the methods provided by the original literature of SAPS II. Thirdly, we performed XGBoost model to analysis the contribution (gain) of each variable to 30-days mortality, at the same time, backward stepwise analysis was processed to select the variable with a threshold of p<0.05 according to the Akaike information criterion (AIC)."

Additionally, the below features are used to calculate SAPS-II:

- Age, GCS
- VITALS: Heart rate, systolic blood pressure, temperature
- FLAGS: ventilation/cpap
- IO: urine output
- LABS: PaO2/FiO2 ratio, blood urea nitrogen, WBC, potassium, sodium, HCO3

Below is an image of which features were selected for both models

```
show_image('img-features')
```

Table 2 Features selected in the conventional logistic regression

	OR_with_CI	p value
(Intercept)	52,913.003 (87.92–33,934,517.782)	< 0.001
Sofa	1.142 (1.106–1.179)	< 0.001
Aniongap_min	1.078 (1.043–1.115)	< 0.001
Creatinine_min	0.676 (0.592–0.767)	< 0.001
Chloride_min	0.98 (0.962–0.999)	0.03393
Hematocrit_min	1.113 (1.053–1.178)	< 0.001
Hemoglobin_min	0.748 (0.623–0.895)	0.00169
Hemoglobin_max	0.926 (0.863–0.992)	0.02993
Lactate_min	1.308 (1.194–1.435)	< 0.001
Potassium_min	1.179 (1.001–1.389)	0.04922
Sodium_max	1.046 (1.019–1.074)	< 0.001
Bun_min	1.033 (1.018–1.048)	< 0.001
Bun_max	0.986 (0.973–0.997)	0.01542
Wbc_min	1.062 (1.036–1.09)	< 0.001
Wbc_max	0.969 (0.952–0.987)	< 0.001
Heartrate_min	0.987 (0.977–0.997)	0.0111
Heartrate_mean	1.022 (1.011–1.033)	< 0.001
Sysbp_min	0.991 (0.984–0.998)	0.00839
Meanbp_min	0.992 (0.985–1)	0.0468
Resprate_mean	1.062 (1.038–1.086)	< 0.001
Tempc_min	0.897 (0.81–0.993)	0.03242
Tempc_max	0.781 (0.698–0.873)	< 0.001
Spo2_mean	0.947 (0.909–0.986)	0.00839
Age	1.029 (1.022–1.035)	< 0.001
Diabetes	0.779 (0.639–0.948)	0.01328
Vent	1.824 (1.48–2.251)	< 0.001

OR odds ratio, *CI* confidence interval, *SOFA* sequential organ failure assessment, *bun* blood urea nitrogen, *wbc* white blood cell, *sysbp* systolic blood pressure, *meanbp* mean blood pressure, *resprate* respiratary rate, *Spo2* oxyhemoglobin saturation, *vent* ventilation, *Max* maximum, *Min* minimum

Table 3 Features selected in the XGboost model

	OR_with_CI	*P*
(Intercept)	493.907 (9.063–27,931.087)	0.00247
Urineoutput	1 (1–1)	< 0.001
Lactate_min	1.401 (1.288–1.527)	< 0.001
Bun_mean	1.018 (1.013–1.023)	< 0.001
Sysbp_min	0.979 (0.974–0.984)	< 0.001
Metastatic_cancer	2.997 (2.217–4.038)	< 0.001
Inr_max	1.058 (1.002–1.115)	0.03709
Age	1.019 (1.013–1.025)	< 0.001
Sodium_max	1.016 (1.001–1.031)	0.03835
Aniongap_max	1.048 (1.026–1.069)	< 0.001
Creatinine_min	0.766 (0.686–0.852)	< 0.001

Features statistics

The target classification of this analysis is to accurately predict if a patient will die within 30 days of their first visit from sepsis. From the subsample of data present in BQ, there are 877 positive records (patients who died within 30 days), and 1443 negative records (patients who did not die within 30 days). All features are numerical, either discrete or continuous.

For the model hyperparameters, not much is described in the paper such as number of epochs, scoring functions, learning rates, etc, so for this paper we will use standard approaches to each, as the hypothesis tested in the paper are less focused on hyperparameter tuning, and more about model selection strengths.

Data preprocessing

Again, not much is described in terms of data preparation and pre-processing - the focus of the paper is on parameter selection and model performance. Because of this, we will be using basic model preprocessing, which will involve the below steps:

1. Download SQL for views, as well as authenticate BigQuery client
2. Drop and reset derived and feature datasets
3. Create views in BQ from raw data tables
4. Create feature-selection views from all derived views and raw data tables
5. Download data from the respective view for all models
6. Drop records that are missing all non-target data
7. Drop non-feature columns
8. Feature normalization using sklearn MinMaxScaler, in order to scale data between 0 and 1 and prevent negative values
9. Create train and test splits for data

```
# 1. Download SQL for views, as well as authenticate BigQuery client

# Read SA credentials for BQ
f = io.BytesIO()
f.write(requests.get(gcloud_url.format(gcloud_ids['sa-creds'])).content)
creds = json.loads(f.getvalue().decode())
# Authorize and close
client = bigquery.Client.from_service_account_info(creds)
f.close()

# Read view SQL files
f = io.BytesIO()
f.write(requests.get(gcloud_url.format(gcloud_ids['mimic-views'])).content)
# Parse from memory into dict
view_data = {}
with zipfile.ZipFile(f, 'r') as zip:
    for file in zip.namelist():
        # Create file name and data from decoded zip bytes
        fname = os.path.basename(file)[:-4].lower() #Last 4 chars are .sql
        # Store into StringIO for pandas to read
        data = zip.read(file).decode()
        view_data[fname] = data
f.close()


# Download model feature SQL files
f = io.BytesIO()
f.write(requests.get(gcloud_url.format(gcloud_ids['model-features'])).content)
# Parse from memory into dict
model_views_data = {}
with zipfile.ZipFile(f, 'r') as zip:
    for file in zip.namelist():
        # Create file name and data from decoded zip bytes
        fname = os.path.basename(file)[:-4].lower() #Last 4 chars are .sql
        # Store into StringIO for pandas to read
        data = zip.read(file).decode()
        model_views_data[fname] = data
f.close()
```

```
# 2. Drop and reset derived and feature datasets

# Create datasets
for dname in [os.environ["BQ_DATASET_DERIVED"], os.environ["BQ_DATASET_FEATURES"]]:
    # Get a list of datasets
    datasets = list(client.list_datasets())  # Make an API request.
    project = client.project
    dataset_id = "{}.{}".format(client.project, dname)

    # Check if the dataset already exists
    dataset_already_exists = dname in [d.dataset_id for d in datasets]

    # Delete if it exists already
    if dataset_already_exists:
        client.delete_dataset(dataset_id, delete_contents=True, not_found_ok=True)  # Make an API request.
        print("Deleted dataset '{}'".format(dataset_id))

    # Create new dataset
    dataset = bigquery.Dataset(dataset_id)
    dataset.location = "US"

    # Send the dataset to the API for creation, with an explicit timeout.
    # Raises google.api_core.exceptions.Conflict if the Dataset already
    # exists within the project.
    dataset = client.create_dataset(dataset, timeout=30)  # Make an API request.
    print("Created dataset {}.{}".format(client.project, dataset.dataset_id))
```

```
 Deleted dataset 'dlh-project-418923.mimiciii_derived'
 Created dataset dlh-project-418923.mimiciii_derived
 Deleted dataset 'dlh-project-418923.models_features'
 Created dataset dlh-project-418923.models_features
```

```
# 3. Create views in BQ from raw data tables

# Create views
for view_name, sql in view_data.items():
    if 'sofa' in view_name: continue #Save sofa for last
    if 'sapsii' in view_name: continue #Save sapsii for last

    # Specify view ID
    view_id = "{}.{}.{}".format(os.environ["GCLOUD_PROJECT"], os.environ["BQ_DATASET_DERIVED"], view_name)
    view = bigquery.Table(view_id)
    view.view_query = sql

    # Make an API request to create the view.
    view = client.create_table(view)
    print(f"Created {view.table_type}: {str(view.reference)}")

# Create sofa view
view_id = "{}.{}.{}".format(os.environ["GCLOUD_PROJECT"], os.environ["BQ_DATASET_DERIVED"], 'sofa')
view = bigquery.Table(view_id)
view.view_query = view_data['sofa']
# Make an API request to create the view.
view = client.create_table(view)
print(f"Created {view.table_type}: {str(view.reference)}")


# Create sapsii view
view_id = "{}.{}.{}".format(os.environ["GCLOUD_PROJECT"], os.environ["BQ_DATASET_DERIVED"], 'sapsii')
view = bigquery.Table(view_id)
view.view_query = view_data['sapsii']
# Make an API request to create the view.
view = client.create_table(view)
print(f"Created {view.table_type}: {str(view.reference)}")
```

```
 Created VIEW: dlh-project-418923.mimiciii_derived.blood_gas_first_day
 Created VIEW: dlh-project-418923.mimiciii_derived.blood_gas_first_day_arterial
 Created VIEW: dlh-project-418923.mimiciii_derived.echo_data
 Created VIEW: dlh-project-418923.mimiciii_derived.elixhauser_ahrq_v37
 Created VIEW: dlh-project-418923.mimiciii_derived.explicit_sepsis
 Created VIEW: dlh-project-418923.mimiciii_derived.gcs_first_day
 Created VIEW: dlh-project-418923.mimiciii_derived.labs_first_day
 Created VIEW: dlh-project-418923.mimiciii_derived.urine_output_first_day
 Created VIEW: dlh-project-418923.mimiciii_derived.ventilation_classifications
 Created VIEW: dlh-project-418923.mimiciii_derived.ventilation_durations
 Created VIEW: dlh-project-418923.mimiciii_derived.vitals_first_day
 Created VIEW: dlh-project-418923.mimiciii_derived.sofa
 Created VIEW: dlh-project-418923.mimiciii_derived.sapsii
```

```
# 4. Create feature-selection views from all derived views and raw data tables

# Create model feature views
for view_name, sql in model_views_data.items():
    # Specify view ID
    view_id = "{}.{}.{}".format(os.environ["GCLOUD_PROJECT"], os.environ["BQ_DATASET_FEATURES"], view_name)
    view = bigquery.Table(view_id)
    view.view_query = sql

    # Make an API request to create the view.
    view = client.create_table(view)
    print(f"Created {view.table_type}: {str(view.reference)}")
```

```
Created VIEW: dlh-project-418923.models_features.logreg_features
Created VIEW: dlh-project-418923.models_features.sapsii_features
Created VIEW: dlh-project-418923.models_features.xgboost_features
```

```
# 5. Download data from the respective view for all models

# Get logistic regression model data
view_id = "{}.{}.{}".format(os.environ["GCLOUD_PROJECT"], os.environ["BQ_DATASET_FEATURES"], os.environ["LOG_REG_FEATURES_VIEW"])
view = client.get_table(view_id)
query_job = client.query(view.view_query)
df_logreg = query_job.result().to_dataframe()
print(f'Successfully downloaded data for logistic regression - shape {df_logreg.shape}')
display(df_logreg.head(3))

# Get XGBoost model data
view_id = "{}.{}.{}".format(os.environ["GCLOUD_PROJECT"], os.environ["BQ_DATASET_FEATURES"], os.environ["XGBOOST_FEATURES_VIEW"])
view = client.get_table(view_id)
query_job = client.query(view.view_query)
df_xgb = query_job.result().to_dataframe()
print(f'Successfully downloaded data for XGBoost - shape {df_xgb.shape}')
display(df_xgb.head(3))

# Get SAPS-II model data
view_id = "{}.{}.{}".format(os.environ["GCLOUD_PROJECT"], os.environ["BQ_DATASET_FEATURES"], os.environ["SAPSII_FEATURES_VIEW"])
view = client.get_table(view_id)
query_job = client.query(view.view_query)
df_sapsii = query_job.result().to_dataframe()
print(f'Successfully downloaded data for SAPS-II - shape {df_sapsii.shape}')
display(df_sapsii.head(3))
```

```
Successfully downloaded data for logistic regression - shape (2320, 31)
```

	subject_id	hadm_id	icustay_id	sofa	aniongap_min	aniongap_max	creatinine_min	ch
0	45248	182072	258959	18	35.0	37.0	2.7	
1	83678	172089	250547	16	31.0	31.0	4.0	
2	32412	155151	219790	9	5.0	12.0	1.6	

3 rows × 31 columns

```
Successfully downloaded data for XGBoost - shape (2222, 15)
```

	subject_id	hadm_id	icustay_id	urineoutput	lactate_min	bun_mean	metastatic_canc
0	7973	122900	201791	490.0	1.2	52.666667	
1	8551	174159	257044	1990.0	NaN	14.000000	
2	15541	115662	270022	1550.0	1.6	18.000000	

```
Successfully downloaded data for SAPS-II - shape (2028, 22)
```

	subject_id	hadm_id	icustay_id	sapsii	sapsii_prob	age_score	hr_score	sysbp_scor
0	28409	167618	205922	42	0.285486	18	4	
1	22337	187366	240086	53	0.530068	18	0	
2	78241	112575	219545	51	0.483852	12	7	1

3 rows × 22 columns

```
# 6. Drop records that are missing all non-target data
# For logistic regression, we cannot have any null values, so any records with null values will be dropped

df_logreg.dropna(how='any', inplace=True)
print(f'Successfully purged null values for logistic regression - shape {df_logreg.shape}')

df_xgb.dropna(thresh=len(df_xgb.columns)-1, inplace=True)
print(f'Successfully purged null values data for XGBoost - shape {df_xgb.shape}')

df_sapsii.dropna(thresh=len(df_sapsii.columns)-1, inplace=True)
print(f'Successfully purged null values data for SAPS-II - shape {df_sapsii.shape}')
```

```
Successfully purged null values for logistic regression - shape (2021, 31)
Successfully purged null values data for XGBoost - shape (2191, 15)
Successfully purged null values data for SAPS-II - shape (1673, 22)
```

```
# 7. Drop non-feature columns
non_feature_cols = ['subject_id', 'hadm_id', 'icustay_id']

df_logreg.drop(non_feature_cols, axis=1, inplace=True)
df_xgb.drop(non_feature_cols, axis=1, inplace=True)
df_sapsii.drop(non_feature_cols, axis=1, inplace=True)
```

```
# 8. Feature normalization using sklearn MinMaxScaler, in order to scale data between 0 and 1 and prevent negative values
# We will not be scaling SAPS-II values since the probability is already calculcated, and that is what will be used for predicting.

from sklearn.preprocessing import MinMaxScaler

# Scale logreg features
scaler = MinMaxScaler()
display(df_logreg.head(3))
df_logreg[df_logreg.columns] = scaler.fit_transform(df_logreg[df_logreg.columns])
display(df_logreg.head(3))

# Scale xgb features
scaler = MinMaxScaler()
display(df_xgb.head(3))
df_xgb[df_xgb.columns] = scaler.fit_transform(df_xgb[df_xgb.columns])
display(df_xgb.head(3))
```

	sofa	aniongap_min	aniongap_max	creatinine_min	chloride_min	hematocrit_min	hemog
0	18	35.0	37.0	2.7	86.0	18.7	
1	16	31.0	31.0	4.0	100.0	11.0	
2	9	5.0	12.0	1.6	115.0	19.4	

3 rows × 28 columns

	sofa	aniongap_min	aniongap_max	creatinine_min	chloride_min	hematocrit_min	h
0	0.857143	0.885714	0.659574	0.230088	0.613984	0.213808	
1	0.761905	0.771429	0.531915	0.345133	0.715950	0.042316	
2	0.428571	0.028571	0.127660	0.132743	0.825200	0.229399	

3 rows × 28 columns

	urineoutput	lactate_min	bun_mean	metastatic_cancer	inr_max	age	sodium_max	ani
0	490.0	1.2	52.666667	0	1.3	74	134.0	
2	1550.0	1.6	18.000000	0	2.4	39	147.0	
3	1128.0	1.6	29.500000	0	NaN	33	138.0	

	urineoutput	lactate_min	bun_mean	metastatic_cancer	inr_max	age	sodium_max
0	0.134575	0.033613	0.232023	0.0	0.021186	0.788732	0.323944
2	0.224291	0.050420	0.071688	0.0	0.067797	0.295775	0.507042
3	0.188574	0.050420	0.124876	0.0	NaN	0.211268	0.380282

```
# 9. Create train and test splits for data

from sklearn.model_selection import train_test_split

# Set test size to 15% of the dataset
test_size = 0.1

# Create logistic regression X and y data
X_logreg = df_logreg.drop('target', axis=1)
y_logreg = df_logreg[['target']]
X_train_logreg, X_test_logreg, y_train_logreg, y_test_logreg = train_test_split(X_logreg, y_logreg, test_size=test_size, random_state=RANDOM

# Create xgboost X and y data
X_xgb = df_xgb.drop('target', axis=1)
y_xgb = df_xgb[['target']]
X_train_xgb, X_test_xgb, y_train_xgb, y_test_xgb = train_test_split(X_xgb, y_xgb, test_size=test_size, random_state=RANDOM_STATE)

# Create SAPSII X, y_true, and y_pred datasets
X_sapsii = df_sapsii.drop('target', axis=1)
y_true_sapsii = df_sapsii[['target']]
y_pred_sapsii = df_sapsii['sapsii_prob']

# Check sizes
print('Logistic regression data shape\n---------------')
print(f'\tX: {X_logreg.shape}')
print(f'\ty: {y_logreg.shape}')
print()
print('XGBoost data shape\n---------------')
print(f'\tX: {X_xgb.shape}')
print(f'\ty: {y_xgb.shape}')
print('SAPS-II data shape\n---------------')
print(f'\ty: {df_sapsii.shape}')
```

```
Logistic regression data shape
---------------
        X: (2021, 27)
        y: (2021, 1)

XGBoost data shape
---------------
        X: (2191, 11)
        y: (2191, 1)
SAPS-II data shape
---------------
        y: (1673, 19)
```

Sizes can vary slightly between the number of valid records between models because the features selected for each model are different, so there can be more or less records with missing values between the two models.

Model

The model includes the model definitation which usually is a class, model training, and other necessary parts.

- Model architecture: layer number/size/type, activation function, etc
- Training objectives: loss function, optimizer, weight of each loss term, etc
- Others: whether the model is pretrained, Monte Carlo simulation for uncertainty analysis, etc
- The code of model should have classes of the model, functions of model training, model validation, etc.
- If your model training is done outside of this notebook, please upload the trained model here and develop a function to load and test it.

Model summary

These There are 3 models included in testing for this paper. They are not pretrained, and will be simply defined below:

1. Linear regression, which will be made using the sklearn library
2. XGBoost model, based on the XGBoost library. This is a type of ensemble method, similar to random forest, but includes features such as efficient regularization, parallel processing, and sparse data handling
3. SAPS-II, which is a probabilistic model and does not require any additional training, as the disease mortality probability is included in the dataset

Models 1 and 3 (Linear Regression and SAPS-II) are considered "traditional" approaches in the paper for predicting ICU mortality rates within timeframes. Often times, a certain threshold is applied to SAPS-II scores, but this paper does not reference one. The goal of the study is to show that more advanced machine learning approaches, such as XGBoost, can outperform traditional methods.

Model architecture

This study focuses heavily on standard approches and evaluation vs newer, less-tested approaches (XGBoost). However, there is little mentioned about model architecture details.

1. For the linear regression model, the formula is very basic and has little tuning involved, so no details are specified or changed.
2. For the XGBoost model, there are quite a few architecture parameters that can be changed, such as:
    - n_estimators - the number of trees in the model
    - max_depth - maximum depth of trees, which helps control how specialized trees are in the ensemble
    - learning rate - the eta parameter, this controls how much each tree contributes to the overall model

    However, within the paper, there are no architecture parameters specified. Instead of trying to give an advantage to the XGBoost model via hyperparameters that were not specified in this paper, we will assume all default values for the model to compare the base-state of the XGBoost model to linear regression and SAPS-II results.
3. For the SAPS-II model, the probability is calculated based on Simplified Acute Physiology Score II, which is a measure of patient severity of illness. This score ranged from 0 and 163, with a predicted mortality of 0 to 1, with 1 being certain death. We will be using this predicted mortality for our model.

Training objectives

1. Linear regression has a non-customizable loss function based on the linear regression equation and we won't be customizing any loss hyperparameters
2. For XGBoost, we will be using the default squarederror loss, as we do not want to make hyperparameter assumptions the authors didn't inform the readers of. This includes other hyperparameters including optimizers, learning steps, minimum loss, etc.
3. SAPS-II is a probabilistic model and has no training involved

Computation requirements

There are limited computational requirements for these models - the unchanged regressors have very limited number of weights, as well as hyperparameters. The number of trees, max depth, and other parameters discussed earlier, when left unchanged, have minimal requirements for the model. Any modern computer can host these models. The largest requirements come from being able to hold the data, which is done via bigquery. The dataframes are not holding much.

Training

```
from sklearn.linear_model import LogisticRegression
import xgboost as xgb

# Set parameters for logistic regression
params_logreg = {}
params_logreg['penalty'] = 'l2'
params_logreg['n_jobs'] = None
params_logreg['random_state'] = RANDOM_STATE

# Create parameters for XGBoost
params_xgb = {}
params_xgb['booster'] = 'gbtree'
params_xgb['objective'] = 'binary:logistic'
params_xgb["eval_metric"] = "auc"
params_xgb['eta'] = 0.1
params_xgb['gamma'] = 0
params_xgb['max_depth'] = 6
params_xgb['min_child_weight']=1
params_xgb['max_delta_step'] = 0
params_xgb['subsample']= 1
params_xgb['colsample_bytree']=1
params_xgb['silent'] = 1
params_xgb['seed'] = 0
params_xgb['base_score'] = 0.5
params_xgb['silent'] = 1
params_xgb['random_state'] = RANDOM_STATE
```

```
# Create models
model_logreg = LogisticRegression(**params_logreg)
model_xgb = xgb.XGBRegressor(**params_xgb)


# Train models
model_logreg.fit(X_train_logreg, y_train_logreg)
model_xgb.fit(X_train_xgb, y_train_xgb)
```

```
/usr/local/lib/python3.10/dist-packages/sklearn/utils/validation.py:1143: DataConversior
  y = column_or_1d(y, warn=True)
/usr/local/lib/python3.10/dist-packages/xgboost/core.py:160: UserWarning: [21:57:12] WAR
Parameters: { "silent" } are not used.

  warnings.warn(smsg, UserWarning)
```

```
XGBRegressor
XGBRegressor(base_score=0.5, booster='gbtree', callbacks=None,
             colsample_bylevel=None, colsample_bynode=None, colsample_bytree=1,
             device=None, early_stopping_rounds=None, enable_categorical=False,
             eta=0.1, eval_metric='auc', feature_types=None, gamma=0,
             grow_policy=None, importance_type=None,
             interaction_constraints=None, learning_rate=None, max_bin=None,
             max_cat_threshold=None, max_cat_to_onehot=None, max_delta_step=0,
             max_depth=6, max_leaves=None, min_child_weight=1, missing=nan,
             monotone_constraints=None, multi_strategy=None, n_estimators=None,
             n_jobs=None, num_parallel_tree=None, ...)
```

Evaluation

For evaluation in this study, the authors used AUROC and DCA curves for comparisons. Additionally, we are going to use standard metrics, including:

- accuracy
- precision
- recall
- support
- F1
- confusion matrix

The reason for these metrics is they can provide a general comparison and results analysis for our 3 models, and can help us understand the performance of each across certain metrics. Understanding accuracy, precision, sensitivity, recall, etc shows different strengths and weaknesses in each model.

For our SAPS-II predictive model, a threshold must be defined for mortality classifications because this is a probabilistic model. The author's do not provide one, so we will assume that anything greater than or equal to probability 0.5 predicts "1", and anything less than 0.5 predicts "0".

```
from sklearn import metrics

# Get probabilistic predictions from each model and use these
predict_logreg = model_logreg.predict_proba(X_test_logreg)[:, 1]
predict_xgb = model_xgb.predict(X_test_xgb)
predict_sapsii = y_pred_sapsii

# Get classifications for each
predict_logreg_clf = np.where(predict_logreg >= 0.5, 1, 0)
predict_xgb_clf = np.where(predict_xgb >= 0.5, 1, 0)
predict_sapsii_clf = np.where(predict_sapsii >= 0.5, 1, 0)


# Get precision, recall, and macro fbeta scores
logreg_metrics = list(metrics.precision_recall_fscore_support(y_test_logreg, predict_logreg_clf, average='macro'))[:-1]
xgb_metrics = list(metrics.precision_recall_fscore_support(y_test_xgb, predict_xgb_clf, average='macro'))[:-1]
sapsii_metrics = list(metrics.precision_recall_fscore_support(y_true_sapsii, predict_sapsii_clf, average='macro'))[:-1]

# Get accuracy scores
logreg_metrics.append(metrics.accuracy_score(y_test_logreg, predict_logreg_clf))
xgb_metrics.append(metrics.accuracy_score(y_test_xgb, predict_xgb_clf))
sapsii_metrics.append(metrics.accuracy_score(y_true_sapsii, predict_sapsii_clf))

# Get confusion matrix
cm_logreg = metrics.confusion_matrix(y_test_logreg, predict_logreg_clf)
cm_xgb = metrics.confusion_matrix(y_test_xgb, predict_xgb_clf)
cm_sapsii = metrics.confusion_matrix(y_true_sapsii, predict_sapsii_clf)
```

```
# Get ROC curves
fpr_logreg, tpr_logreg, _ = metrics.roc_curve(y_test_logreg, predict_logreg)
fpr_xgb, tpr_xgb, _ = metrics.roc_curve(y_test_xgb, predict_xgb)
fpr_sapsii, tpr_sapsii, _ = metrics.roc_curve(y_true_sapsii, y_pred_sapsii)

# Get AUC scores
logreg_metrics.append(metrics.roc_auc_score(y_test_logreg, predict_logreg))
xgb_metrics.append(metrics.roc_auc_score(y_test_xgb, predict_xgb))
sapsii_metrics.append(metrics.roc_auc_score(y_true_sapsii, y_pred_sapsii))


# Get the DCA for the models

# Calculate the net benefit based on a given threshold
def net_benefit(tp, fp, tn, fn, threshold):
    benefit = tp - (fp * threshold / (1 - threshold))
    total = tp + fn  # Total number of actual positives
    return benefit / total

# DCA
def decision_curve_analysis(y_true, y_prob):
    thresholds = np.linspace(0.01, 0.99, 100)
    net_benefits = []
    for threshold in thresholds:
      y_pred = np.where(y_prob >= threshold, 1, 0)
      tn, fp, fn, tp = metrics.confusion_matrix(y_true, y_pred).ravel()
      nb = net_benefit(tp, fp, tn, fn, threshold)
      net_benefits.append(nb)

    return thresholds, net_benefits
```

Results

Looking at results, the XGBoost model performs slightly better than the logistic regression model. Through research, the deterministic nature of XGBoost can usually lead to repeated results, although results have varied slightly in the past.

For `precision`, `recall`, `f1`, `accuracy`, and `AUC`, XGBoost performs the best, with `recall`, `f1`, and `accuracy` being a sizable difference, and `precision` and `AUC` being similar to logistic regression.

For the SAPS-II model, all metrics were signficicantly outperformed by the Logistic Regression model, and the XGBoost model. The ROC curve for the SAPS-II model also indicated inferior predictive performance compared to the other models.

```
# Display results for metric scores as a dataframe
# Create dict out of these
data = {'metric': ['precision', 'recall', 'f1', 'accuracy', 'AUC'], 'LogReg': logreg_metrics, 'XGBoost': xgb_metrics, 'SAPS-II': sapsii_metr
df_metrics = pd.DataFrame(data).round(4)
df_metrics.set_index('metric', drop=True, inplace=True)

# Display data
display(df_metrics)
```

metric	LogReg	XGBoost	SAPS-II
precision	0.7594	0.6823	0.5172
recall	0.6852	0.6554	0.5277
f1	0.6909	0.6608	0.4849
accuracy	0.7389	0.7045	0.5601
AUC	0.7518	0.6938	0.5236

Next steps: View recommended plots

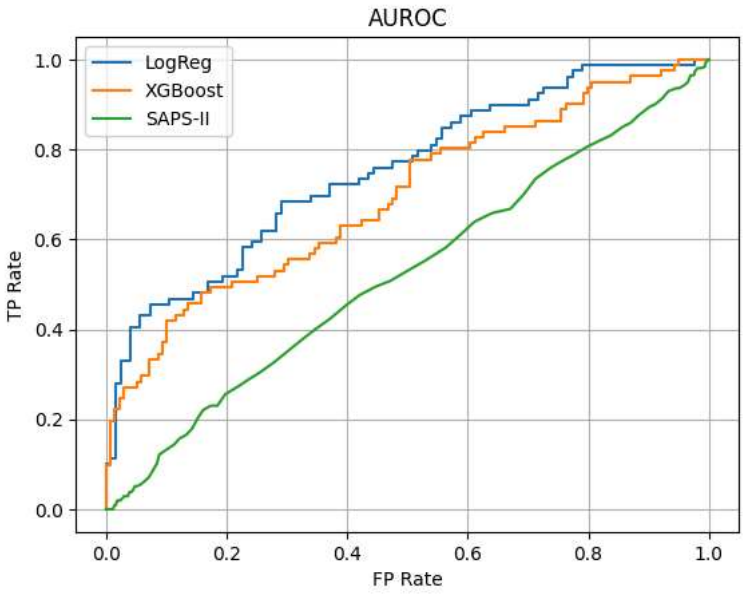
ROC Curve

Looking at the ROC curves below for each model type, we can see the difference at each threshold between models. The XGBoost model indicates better performance at lower thresholds, but is evenly matched, and sometimes even outperformed, by the Logistic Regression model,

at mid-to-higher thresholds. The SAPS-II model indicated average-to-poor predictive performance, providing a minimal ROC curve more closely matching a linear increase.

```
# Plot AUROC
plt.figure()
plt.plot(fpr_logreg, tpr_logreg, label='LogReg')
plt.plot(fpr_xgb, tpr_xgb, label='XGBoost')
plt.plot(fpr_sapsii, tpr_sapsii, label='SAPS-II')

# Create plot
plt.title('AUROC')
plt.xlabel('FP Rate')
plt.ylabel('TP Rate')
plt.legend()
plt.grid()
plt.show()
```



DCA Scores

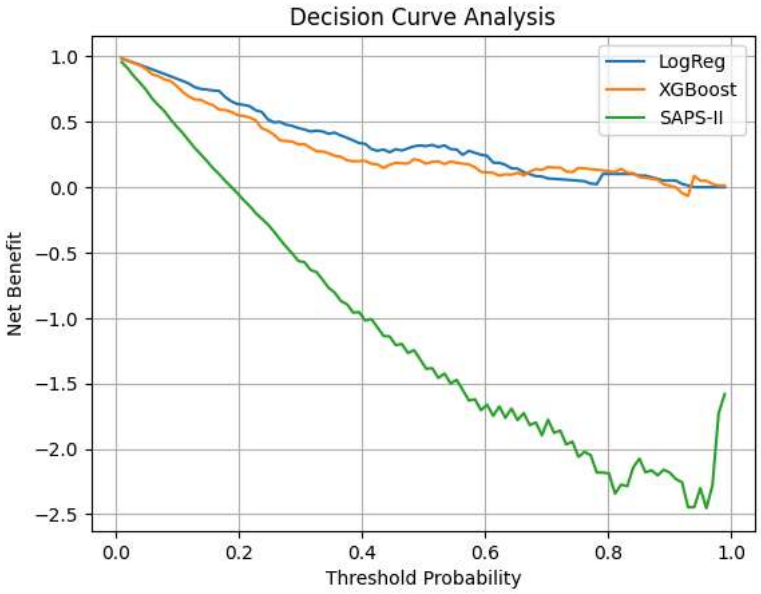
DCA, or decision-curve-analysis, is used to compare clinical usefulness and net benefits between each model. The analysis helps in understanding tradeoffs between the benefit of true positive predictions, and the harm of false positive predictions within a clinical context.

From the DCA curve below, we can see that the Logistic Regression model and XGBoost model perform similarly at each threshold, with some thresholds showing the XGBoost model outperforming. The SAPS-II graph has poor results here, showing low benefit at each threshold.

```
# Actually calculate the DCA scores
thresholds_logreg, net_benefits_logreg = decision_curve_analysis(y_test_logreg, predict_logreg)
thresholds_xgb, net_benefits_xgb = decision_curve_analysis(y_test_xgb, predict_xgb)
thresholds_sapsii, net_benefits_sapsii = decision_curve_analysis(y_true_sapsii, y_pred_sapsii)

plt.figure()
plt.plot(thresholds_logreg, net_benefits_logreg, label='LogReg')
plt.plot(thresholds_xgb, net_benefits_xgb, label='XGBoost')
plt.plot(thresholds_sapsii, net_benefits_sapsii, label='SAPS-II')

plt.xlabel('Threshold Probability')
plt.ylabel('Net Benefit')
plt.title('Decision Curve Analysis')
plt.legend()
plt.grid()
plt.show()
```



Model comparison

Comparing the models created in my subsample study to the actual full study, results performed similarly for the Logistic Regression and XGBoost models. For the SAPS-II model, results were not as good, but this could be due to multiple factors that will be brought up in the "Discussion" section.

The actual values for AUC, the ROC curve, and the DCA curves were not as impressive as the actual study - from the graphs below, I will compare the models I created, vs the models from the study. While trends were similar, results were simply better from the study. This could be due to multiple factors, such as the sample selection for my case study failed to capture certain features, or parameter selection differed due to unwritten differences the authors did not provide.

The authors do not provide other summary metrics such as accuracy, precision, recall, or F1-scores, but we can see similar trends matching in this study vs from L. et al. [1].

Comparing ROC curves

We can see similar differences in the Logistics Regression model vs the XGBoost model. The SAPS-II model provided outperforms from L. et al. [1] vs in this sub-study.

```
print('From L. et al. [1]')
show_image('graph-roc', resize=True)
print('From this study')
show_image('my-roc', resize=True)
```

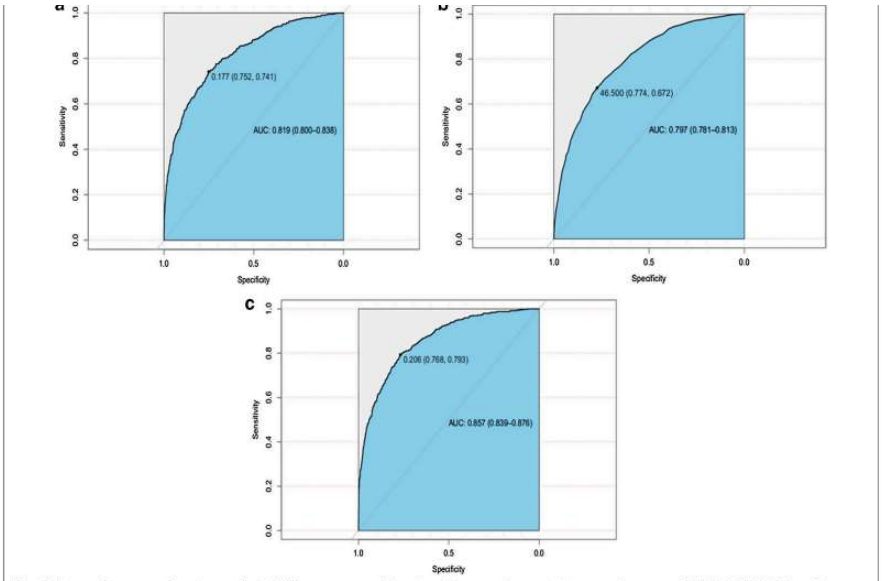
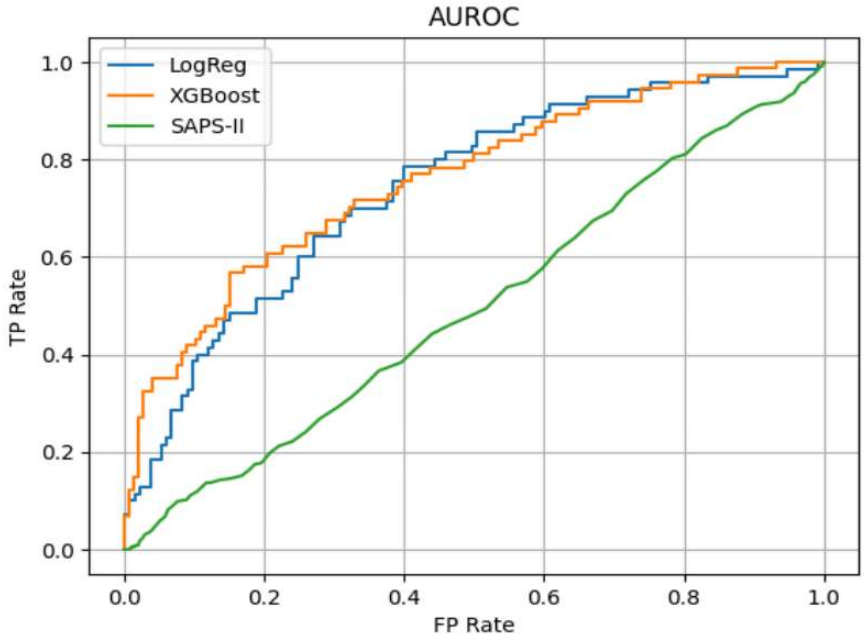


Fig. 4 The receiver operating characteristic (ROC) curves. **a** traditional logistic regression model, area under curves (AUC) is 0.819 [95% confidence interval (CI); 0.800–0.838]; **b** SAPS-II score model, AUC is 0.797 [0.781–0.813]; **c** XGboost model, AUC is 0.857 [0.839–0.876], the best performance of the models was the XGboost model

From this study



Comparing DCA curves

```
print('From L. et al. [1]')
show_image('graph-dca', resize=True)
print('From this study')
show_image('my-dca', resize=True)
```

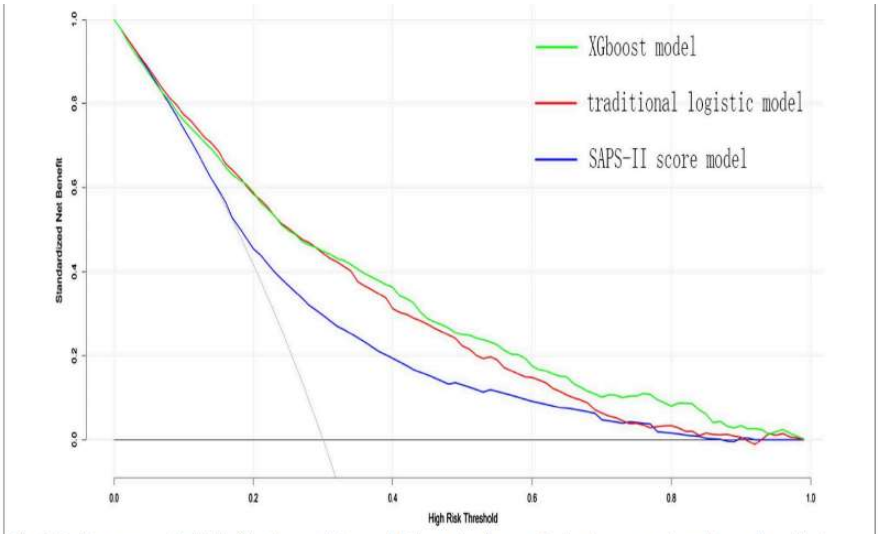
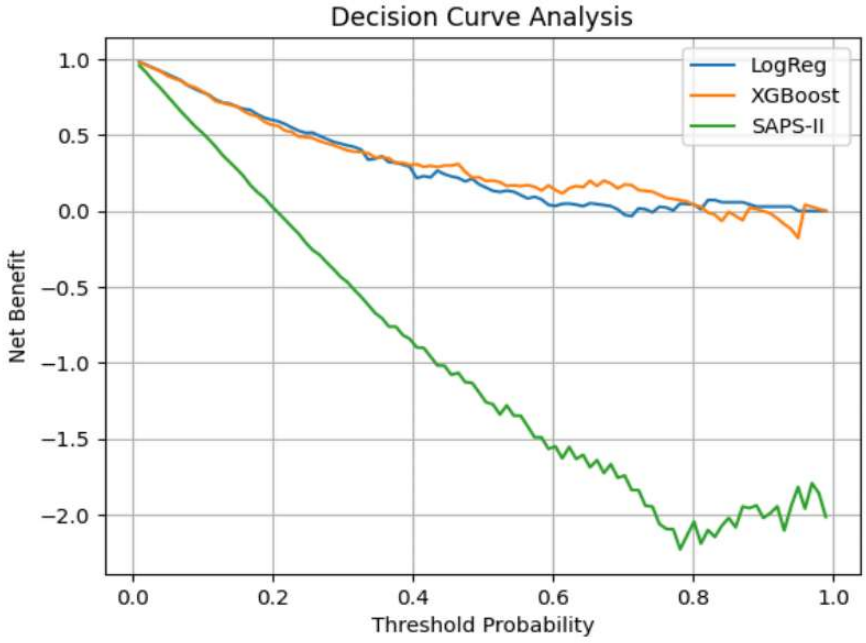


Fig. 5 Decision curve analysis (DCA) of the three prediction models. The net benefit curves for the three prognostic models are shown. *X*-axis indicates the threshold probability for critical care outcome and *Y*-axis indicates the net benefit. Solid green line = XGboost model, solid red line = traditional logistic model, solid blue line = SAPS-II score mode. The preferred model is the XGboost model, the net benefit of which was larger over the range of traditional logistic model and SAPS-II score model

From this study



Nomogram

The authors also provide a nomogram, which helps us see how important each feature is to sepsis mortality rates in this study. In the chart below, we can see how each feature is relevant for scoring in mortality predictions, and assign point values to different features. This is helpful in understanding how each feature plays a part in mortality scoring.