capstone

May 9, 2021

1 HDAT Capstone Project - Josh Bryden

1.1 Research Question 1 - Mortality prediction in the ICU:

Task - The task is to build a predictive algorithm using the techniques we learned in this course

Objective - To assess the role of machine learning algorithms for predicting mortality by using the MIMIC-II dataset

Question - Is it possible to accurately predict mortality based on data from the first 24 hours in ICU?

Study population - MIMIC-II dataset

1.2 Research Question 2 - Weekend Effect in the ICU

Task - The task is to investigate whether admission to ICU at the weekend increases the risk of ICU mortality

Objective - To develop a statistical model to estimate the effect of weekend admission to ICU on the risk of mortality.

Question - Does admission to ICU over the weekend increase the risk of mortality?

Study population - MIMIC-II dataset ICU mortality is commonly researched across health departments, health districts and nations and is seen as a measure of the overall healthcare system effectiveness (Khan & Ridley, 2014). In the United Kingdom, approximately 120,000 critical patients enter are admitted to an ICU either in England, Wales and Northern Ireland, of which 77% survive (Khan & Ridley, 2014). In the United States, approximately four million patients are admitted to an ICU nationally, with mortality rates reported to be in the 8-19% range dependant on hospital, resulting in approximately half a million deaths annually (Wu et al, 2002, Angus et al, 1996). With the pool of patients that are admitted to the ICU usually requiring complex models of care provided by various specialists, patients are exposed to adverse health outcomes. Giraud et al, 1993 found that in two ICU departments in France that iatrogenic complications (whereby complications are caused by medical treatments) increased morbidity and mortality rates and were often caused by a stretched staff workload. Iatrogenic complications coupled with high costs associated with ICU operations in both private healthcare models (United States) and in public healthcare models (Australia), (Hicks et al, 2019). As such, research into ICU morbidly and mortality rates are of great importance to researchers and healthcare providers to reduce the overall cost on the healthcare system, both from a costs and strain on resources point of view. The present study aims to address these issues by investigating mortality rates using data obtained from a patients first 24 hours in an ICU and investigating what is known throughout the present-day literature as the 'weekend effect', whereby patients admitted to an ICU on the weekend are associated with increased mortality (Faust et al, 2019). The dataset used in this study is the Medical Information Mart for Intensive Care II (MIMIC-II) which is a large and deidentified dataset containing data from 40,000 patients who were admitted to the ICU units of the Beth Israel Deaconess Medical Centre between 2001 and 2012. Components of the dataset will be used to build a predictive model using data from the first 24 hours after ICU admission to predict mortality inside the ICU. Such a model could be used to highlight the patients in most need of care and assign them a priority status in an effort to lower mortality rates across the board. Furthermore, using a similar dataset, we will build a predictive/ statistical model to examine the effects (if any) of the weekend effect in the MIMIC-II dataset. Such a model can be used to inform hospital policy through staffing in order to increase patient outcomes, such a model can also be applied to other hospital settings to inform decision making in a different hospital environment.

2 Imports

[114]:
```python
import pandas as pd
pd.set_option('display.max_columns', None) # show all columns
pd.options.display.float_format = '{:.2f}'.format # show only 2 decimal places␣
↪on output
import numpy as np
import seaborn as sns
sns.set_style("darkgrid") # sets seaborn plot style guide
import matplotlib.pyplot as plt
from imblearn.over_sampling import SMOTE
import lime
import lime.lime_tabular
import statsmodels.api as sm

# Import train test split function:
from sklearn.model_selection import train_test_split
# Import the standard scaler function:
from sklearn.preprocessing import StandardScaler
# Import the logistic regression model:
from sklearn.linear_model import LogisticRegression
# Import the pipeline function:
from sklearn.pipeline import Pipeline
# Import the grid search CV class:
from sklearn.model_selection import GridSearchCV
# Import the RandomizesearchCV function:
from sklearn.model_selection import RandomizedSearchCV
# Import RandomForestClassifier
from sklearn.ensemble import RandomForestClassifier
```

```python
# Import tf.keras
import tensorflow.keras
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense, Dropout, Activation
from tensorflow.keras.layers import Flatten
from tensorflow.keras import backend as K
from tensorflow.keras import optimizers
# Import pacakges to link sklearn with tf.keras
from tensorflow.keras.wrappers.scikit_learn import KerasClassifier

# Import the accuracy_score function:
from sklearn.metrics import accuracy_score
# Import the classification report function:
from sklearn.metrics import classification_report
# Import the confusion matrix function:
from sklearn.metrics import confusion_matrix

# Set global seed to make sure results are reproducible:
np.random.seed(1111)
# Set tensorflow seed
from tensorflow import set_random_seed
set_random_seed(1111)
```

The MIMIC-II dataset was not collected for the aims of research and is a collection of data stemming from two electronic medical record (EMR) systems. As such there is a large amount of data with missing values and/or vast inaccuracies. Throughout the analysis, decisions had to be made surrounding methods to deal with these inaccuracies and missing data. The MIMIC-II dataset is a combination of multiple tables outlines patients, their characteristics, a patients hospital stay, their ICU stay and a multitude of additional variables that were recorded such as lab results, vital measurements, urine output, risk assessment measures and pharmaceuticals administered. The lowest discrete time interval that MIMIC-II establishes is hourly measurements. A table called pt_stay_hr was created which is an amalgamation of tables from MIMIC-II that captures the patient ID, their hospital stay ID, admission and discharge dates/times and an hour and day column that tracks hour by hour a patients stay in the ICU. This table begins for all patients with an ICU stay ID at 24 hours before admission as some patients have transferred into the ICU from other hospital wards and hence will have measurements taken before ICU entry. This table was merged with the MIMIC-II table called 'patients' where the variables 'subject_ID' (patient ID), gender and date of birth were captured. These variables were merged by 'subject_ID' with the 'pt_stay_hr' table to create the 'master' table.

2.1 Loading in datasets, merging and cleaning

[115]:
```
# load in pt_stay_hr as the building block / master table
pt_stay_hr = pd.read_csv('mimic_data/pt_stay_hr.csv')
pt_stay_hr.head()
```

[115]:
```
   icustay_id  hadm_id  subject_id               intime              outtime  \
0      200001   152234       55973  2181-11-25 19:06:12  2181-11-28 20:59:25
1      200001   152234       55973  2181-11-25 19:06:12  2181-11-28 20:59:25
2      200001   152234       55973  2181-11-25 19:06:12  2181-11-28 20:59:25
3      200001   152234       55973  2181-11-25 19:06:12  2181-11-28 20:59:25
4      200001   152234       55973  2181-11-25 19:06:12  2181-11-28 20:59:25

            starttime              endtime     hr   dy
0  2181-11-24 19:06:12  2181-11-24 20:06:12 -24.00 0.00
1  2181-11-24 20:06:12  2181-11-24 21:06:12 -23.00 0.00
2  2181-11-24 21:06:12  2181-11-24 22:06:12 -22.00 0.00
3  2181-11-24 22:06:12  2181-11-24 23:06:12 -21.00 0.00
4  2181-11-24 23:06:12  2181-11-25 00:06:12 -20.00 0.00
```

[116]:
```
patients = pd.read_csv('mimic_data/patients.csv') # https://mimic.physionet.org/
 ↪mimictables/patients/
# Table purpose: Defines each SUBJECT_ID in the database, i.e. defines a single␣
 ↪patient
# Links to: ADMISSIONS on SUBJECT_ID, ICUSTAYS on SUBJECT_ID
patients.head()
```

[116]:
```
   row_id  subject_id gender                  dob                  dod  \
0     234         249      F  2075-03-13 00:00:00                  NaN
1     235         250      F  2164-12-27 00:00:00  2188-11-22 00:00:00
2     236         251      M  2090-03-15 00:00:00                  NaN
3     237         252      M  2078-03-06 00:00:00                  NaN
4     238         253      F  2089-11-26 00:00:00                  NaN

              dod_hosp dod_ssn  expire_flag
0                  NaN     NaN            0
1  2188-11-22 00:00:00     NaN            1
2                  NaN     NaN            0
3                  NaN     NaN            0
4                  NaN     NaN            0
```

[117]:
```
# Select only columns of interest from patients
patients = patients[['subject_id','gender','dob']]
patients.head()
```

[117]:
```
   subject_id gender                  dob
0         249      F  2075-03-13 00:00:00
```

```
1         250      F  2164-12-27 00:00:00
2         251      M  2090-03-15 00:00:00
3         252      M  2078-03-06 00:00:00
4         253      F  2089-11-26 00:00:00
```

[118]:
```python
# Merge pt_stay_hr and patients to master table on subject_id

master = pd.merge(pt_stay_hr, patients, on='subject_id')
master.head()
# intime + outtime = ICU in and out times
# hr starts from -24 = 24 hrs before admission
# dy days in ICU
# starttime and endtime = start and end of each hr interval
```

[118]:
```
   icustay_id  hadm_id  subject_id              intime             outtime  \
0      200001   152234       55973 2181-11-25 19:06:12 2181-11-28 20:59:25
1      200001   152234       55973 2181-11-25 19:06:12 2181-11-28 20:59:25
2      200001   152234       55973 2181-11-25 19:06:12 2181-11-28 20:59:25
3      200001   152234       55973 2181-11-25 19:06:12 2181-11-28 20:59:25
4      200001   152234       55973 2181-11-25 19:06:12 2181-11-28 20:59:25

            starttime             endtime     hr   dy gender  \
0 2181-11-24 19:06:12 2181-11-24 20:06:12 -24.00 0.00      F
1 2181-11-24 20:06:12 2181-11-24 21:06:12 -23.00 0.00      F
2 2181-11-24 21:06:12 2181-11-24 22:06:12 -22.00 0.00      F
3 2181-11-24 22:06:12 2181-11-24 23:06:12 -21.00 0.00      F
4 2181-11-24 23:06:12 2181-11-25 00:06:12 -20.00 0.00      F

                  dob
0 2120-10-31 00:00:00
1 2120-10-31 00:00:00
2 2120-10-31 00:00:00
3 2120-10-31 00:00:00
4 2120-10-31 00:00:00
```

The master table was then merged with the 'pt_icu_outcome' table that stored the patients, age, their ICU length of stay and a flag variable to capture if the patient died in the ICU or not. These variables of interest were left joined with the master table using the 'icustay_ID' as the linkage key.

[119]:
```python
# Load pt_icu_outcome dataset
pt_icu_outcome = pd.read_csv('mimic_data/pt_icu_outcome.csv')
pt_icu_outcome.head()
```

[119]:
```
   row_id  subject_id                 dob  hadm_id           admittime  \
0       1           2 2138-07-17 00:00:00   163353 2138-07-17 19:04:00
1       2           3 2025-04-11 00:00:00   145834 2101-10-20 19:08:00
```

```
2       3           4  2143-05-12 00:00:00   185777  2191-03-16 00:28:00
3       4           5  2103-02-02 00:00:00   178980  2103-02-02 04:31:00
4       5           6  2109-06-21 00:00:00   107064  2175-05-30 07:15:00

             dischtime  icustay_id  age_years              intime  \
0  2138-07-21 15:48:00      243653       0.00  2138-07-17 21:20:07
1  2101-10-31 13:58:00      211552      76.00  2101-10-20 19:10:11
2  2191-03-23 18:41:00      294638      47.00  2191-03-16 00:29:31
3  2103-02-04 12:15:00      214757       0.00  2103-02-02 06:04:24
4  2175-06-15 16:00:00      228232      65.00  2175-05-30 21:30:54

               outtime  los hosp_deathtime  icu_expire_flag  \
0  2138-07-17 23:32:21 0.09            NaN                0
1  2101-10-26 20:43:09 6.06            NaN                0
2  2191-03-17 16:46:31 1.68            NaN                0
3  2103-02-02 08:06:00 0.08            NaN                0
4  2175-06-03 13:39:54 3.67            NaN                0

   hospital_expire_flag                  dod  expire_flag  ttd_days
0                  0.00                  NaN            0       NaN
1                  0.00  2102-06-14 00:00:00            1    236.00
2                  0.00                  NaN            0       NaN
3                  0.00                  NaN            0       NaN
4                  0.00                  NaN            0       NaN
```

[120]:

```
# Selecting only columns of interest from pt_icu
pt_icu_outcome =␣
 ↪pt_icu_outcome[['icustay_id','age_years','los','icu_expire_flag']]
pt_icu_outcome.head()
```

[120]:

```
   icustay_id  age_years  los  icu_expire_flag
0      243653       0.00 0.09                0
1      211552      76.00 6.06                0
2      294638      47.00 1.68                0
3      214757       0.00 0.08                0
4      228232      65.00 3.67                0
```

[121]:

```
# Left join the master table with our selected variables from pt_icu_outcome on␣
 ↪the icustay_id
master = pd.merge(master, pt_icu_outcome, on='icustay_id', how='left')
master.head()
```

[121]:

```
   icustay_id  hadm_id  subject_id               intime              outtime  \
0      200001   152234       55973  2181-11-25 19:06:12  2181-11-28 20:59:25
1      200001   152234       55973  2181-11-25 19:06:12  2181-11-28 20:59:25
2      200001   152234       55973  2181-11-25 19:06:12  2181-11-28 20:59:25
3      200001   152234       55973  2181-11-25 19:06:12  2181-11-28 20:59:25
```

```
4      200001   152234       55973  2181-11-25 19:06:12  2181-11-28 20:59:25

             starttime             endtime     hr   dy gender  \
0  2181-11-24 19:06:12  2181-11-24 20:06:12 -24.00 0.00      F
1  2181-11-24 20:06:12  2181-11-24 21:06:12 -23.00 0.00      F
2  2181-11-24 21:06:12  2181-11-24 22:06:12 -22.00 0.00      F
3  2181-11-24 22:06:12  2181-11-24 23:06:12 -21.00 0.00      F
4  2181-11-24 23:06:12  2181-11-25 00:06:12 -20.00 0.00      F

                   dob  age_years  los  icu_expire_flag
0  2120-10-31 00:00:00      61.00 3.08                0
1  2120-10-31 00:00:00      61.00 3.08                0
2  2120-10-31 00:00:00      61.00 3.08                0
3  2120-10-31 00:00:00      61.00 3.08                0
4  2120-10-31 00:00:00      61.00 3.08                0
```

One of the measurements captured by the EMR in MIMIC-II was the Glasgow Coma Scale, (GCS) which measures a patient's level of consciousness, the table 'gcs' contains this variable and the hour during the ICU stay in which the measurement was taken. The GCS variable was left joined to the master table on the patient's ICU ID. As not all patients had GCS values for each hour whilst in the ICU, missing values were filled with the median of GCS scores as missing values cannot be present in both the machine learning and statistical models. The median allowed for outlier values to be captured and retain a semi normal distribution of scores.

[122]:
```
# Load in gcs_hourly --> Glasgow Coma Score
gcs_hourly = pd.read_csv('mimic_data/gcs_hourly.csv')
gcs_hourly.head()
```

[122]:

	icustay_id	hr	gcs	gcseyes	gcsmotor	gcsverbal	endotrachflag
0	200001	0	15	4.00	6.00	5.00	0
1	200001	4	15	4.00	6.00	5.00	0
2	200001	11	15	4.00	6.00	5.00	0
3	200001	13	15	4.00	6.00	5.00	0
4	200001	16	14	3.00	6.00	5.00	0

[123]:
```
# Select variables of interest --> gcs (overall score) and endotrachflag␣
↪(endotracheal tube) + hr and icustay_id for linkage
gcs_hourly = gcs_hourly[['icustay_id', 'hr', 'gcs', 'endotrachflag']]
gcs_hourly.head()
```

[123]:

	icustay_id	hr	gcs	endotrachflag
0	200001	0	15	0
1	200001	4	15	0
2	200001	11	15	0
3	200001	13	15	0
4	200001	16	14	0

[124]:
```
master = pd.merge(master, gcs_hourly, how='left', on=['icustay_id','hr'])
# if no gcs score recorded then replace with the median gcs score of the cohort␣
↪--> as more patients had higher values (indicating they were awake -  we␣
↪want the median)
master['gcs'] = master['gcs'].fillna(master['gcs'].median())
# if no endotrachflag then 0 as no endotrach present for purposes here
master['endotrachflag'] = master['endotrachflag'].fillna(0)
master.head()
```

[124]:
```
   icustay_id  hadm_id  subject_id               intime              outtime  \
0      200001   152234       55973  2181-11-25 19:06:12  2181-11-28 20:59:25
1      200001   152234       55973  2181-11-25 19:06:12  2181-11-28 20:59:25
2      200001   152234       55973  2181-11-25 19:06:12  2181-11-28 20:59:25
3      200001   152234       55973  2181-11-25 19:06:12  2181-11-28 20:59:25
4      200001   152234       55973  2181-11-25 19:06:12  2181-11-28 20:59:25

             starttime              endtime     hr   dy gender  \
0  2181-11-24 19:06:12  2181-11-24 20:06:12 -24.00 0.00      F
1  2181-11-24 20:06:12  2181-11-24 21:06:12 -23.00 0.00      F
2  2181-11-24 21:06:12  2181-11-24 22:06:12 -22.00 0.00      F
3  2181-11-24 22:06:12  2181-11-24 23:06:12 -21.00 0.00      F
4  2181-11-24 23:06:12  2181-11-25 00:06:12 -20.00 0.00      F

                   dob  age_years  los  icu_expire_flag   gcs  endotrachflag
0  2120-10-31 00:00:00      61.00 3.08                0 13.00           0.00
1  2120-10-31 00:00:00      61.00 3.08                0 13.00           0.00
2  2120-10-31 00:00:00      61.00 3.08                0 13.00           0.00
3  2120-10-31 00:00:00      61.00 3.08                0 13.00           0.00
4  2120-10-31 00:00:00      61.00 3.08                0 13.00           0.00
```

Vital measurements such as heart rate, SpO2 and mean arterial pressure (MAP) were also captured by the EMR in MIMIC-II. These variables were used as they provided an overview of physiological function, specifically the MAP encompasses both systolic and diastolic blood pressures in one variable, thus is useful in both models being produced. These variables were captured in the table 'vitals_hourly' and were left joined to the master table on the ICU ID and the hour columns, median values for each variable were used to fill in missing/ blank data as this was required for the models, whilst still ensuring outliers are captured and not influencing on our 'synthetic' values.

[125]:
```
# load in vitals_hourly dataset
vitals_hourly = pd.read_csv('mimic_data/vitals_hourly.csv')
vitals_hourly.head()
```

[125]:
```
   icustay_id  hr   spo2  fio2  temperature  resprate  heartrate  sysbp  \
0      200001   1  98.00   NaN          NaN     18.00     108.00 113.00
1      200001   2  98.00   NaN          NaN     27.00     110.00 116.00
2      200001   3  99.80   NaN        37.67     21.00     102.00 102.00
3      200001   4  94.00   NaN          NaN     19.00     108.00 103.00
```

```
4      200001   5 100.00 35.00         NaN     28.00     104.00 106.00

   diasbp  glucose  meanarterialpressure
0   68.00      NaN                 79.00
1   68.00   118.00                 79.00
2   61.00      NaN                 71.00
3   58.00      NaN                 69.00
4   62.00      NaN                 73.00
```

[126]:
```python
# Extract variables of interest --> spo2, heartrate, meanarterialpressure +␣
↪icustay_id and hr for linkage
# spo2 chosen as there is little NA values over fio2 --> constantly monitored,␣
↪low values are a sign of a serious failure in the respiratory system
# heartrate can be an estimate of cardiovascular function and overall␣
↪physiological function
# mean arterial pressure (MAP) incorporates systolic and diastolic blood␣
↪pressures into one measure and hence is quite useful for machine learning

vitals_hourly = vitals_hourly[['icustay_id', 'hr', 'spo2', 'heartrate',␣
↪'meanarterialpressure']]
vitals_hourly.head()
```

[126]:
```
   icustay_id  hr   spo2  heartrate  meanarterialpressure
0      200001   1  98.00     108.00                 79.00
1      200001   2  98.00     110.00                 79.00
2      200001   3  99.80     102.00                 71.00
3      200001   4  94.00     108.00                 69.00
4      200001   5 100.00     104.00                 73.00
```

[127]:
```python
# merge with master
master = pd.merge(master, vitals_hourly, how='left', on=['icustay_id','hr'])
# if no heart rate recorded then give median heart rate
master['heartrate'] = master['heartrate'].fillna(master['heartrate'].median())
# if no spo2 recorded then give median spo2 from cohort
master['spo2'] = master['spo2'].fillna(master['spo2'].median())
# if no MAP recorded, then give cohort median MAP
master['meanarterialpressure'] = master['meanarterialpressure'].
↪fillna(master['meanarterialpressure'].median())
master.head()
```

[127]:
```
   icustay_id  hadm_id  subject_id              intime             outtime  \
0      200001   152234       55973 2181-11-25 19:06:12 2181-11-28 20:59:25
1      200001   152234       55973 2181-11-25 19:06:12 2181-11-28 20:59:25
2      200001   152234       55973 2181-11-25 19:06:12 2181-11-28 20:59:25
3      200001   152234       55973 2181-11-25 19:06:12 2181-11-28 20:59:25
4      200001   152234       55973 2181-11-25 19:06:12 2181-11-28 20:59:25
```

```
             starttime             endtime     hr   dy gender  \
0  2181-11-24 19:06:12  2181-11-24 20:06:12 -24.00 0.00      F
1  2181-11-24 20:06:12  2181-11-24 21:06:12 -23.00 0.00      F
2  2181-11-24 21:06:12  2181-11-24 22:06:12 -22.00 0.00      F
3  2181-11-24 22:06:12  2181-11-24 23:06:12 -21.00 0.00      F
4  2181-11-24 23:06:12  2181-11-25 00:06:12 -20.00 0.00      F

                   dob  age_years  los  icu_expire_flag   gcs  endotrachflag  \
0  2120-10-31 00:00:00      61.00 3.08                0 13.00           0.00
1  2120-10-31 00:00:00      61.00 3.08                0 13.00           0.00
2  2120-10-31 00:00:00      61.00 3.08                0 13.00           0.00
3  2120-10-31 00:00:00      61.00 3.08                0 13.00           0.00
4  2120-10-31 00:00:00      61.00 3.08                0 13.00           0.00

   spo2  heartrate  meanarterialpressure
0 98.00      92.00                 77.00
1 98.00      92.00                 77.00
2 98.00      92.00                 77.00
3 98.00      92.00                 77.00
4 98.00      92.00                 77.00
```

Urine output aggregated by hour and patient was captured in the table 'output_hourly' and was left joined on hour and ICU ID to the master table. This variable was included as reduced urine output has been shown to show reduced physiological function and is a known clinical sign for mortality (Hui et al, 2014).

[128]:
```
# load in urine output
output_hourly = pd.read_csv('mimic_data/output_hourly.csv')
output_hourly.head()
```

[128]:
```
   icustay_id  hr  urineoutput
0      200001   2       250.00
1      200001  25        60.00
2      200001  62        50.00
3      200003   0       230.00
4      200003   2         0.00
```

[129]:
```
# merge with master
master = pd.merge(master, output_hourly, how='left', on=['icustay_id','hr'])
# if no urine output recorded then replace with 0 ml
master['urineoutput'] = master['urineoutput'].fillna(0)
master.head()
```

[129]:
```
   icustay_id  hadm_id  subject_id              intime             outtime  \
0      200001   152234       55973 2181-11-25 19:06:12 2181-11-28 20:59:25
1      200001   152234       55973 2181-11-25 19:06:12 2181-11-28 20:59:25
2      200001   152234       55973 2181-11-25 19:06:12 2181-11-28 20:59:25
```

```
3     200001   152234       55973  2181-11-25 19:06:12  2181-11-28 20:59:25
4     200001   152234       55973  2181-11-25 19:06:12  2181-11-28 20:59:25

            starttime             endtime     hr   dy gender  \
0  2181-11-24 19:06:12  2181-11-24 20:06:12 -24.00 0.00      F
1  2181-11-24 20:06:12  2181-11-24 21:06:12 -23.00 0.00      F
2  2181-11-24 21:06:12  2181-11-24 22:06:12 -22.00 0.00      F
3  2181-11-24 22:06:12  2181-11-24 23:06:12 -21.00 0.00      F
4  2181-11-24 23:06:12  2181-11-25 00:06:12 -20.00 0.00      F

                  dob  age_years  los  icu_expire_flag   gcs  endotrachflag  \
0  2120-10-31 00:00:00      61.00 3.08                0 13.00           0.00
1  2120-10-31 00:00:00      61.00 3.08                0 13.00           0.00
2  2120-10-31 00:00:00      61.00 3.08                0 13.00           0.00
3  2120-10-31 00:00:00      61.00 3.08                0 13.00           0.00
4  2120-10-31 00:00:00      61.00 3.08                0 13.00           0.00

   spo2  heartrate  meanarterialpressure  urineoutput
0 98.00      92.00                 77.00         0.00
1 98.00      92.00                 77.00         0.00
2 98.00      92.00                 77.00         0.00
3 98.00      92.00                 77.00         0.00
4 98.00      92.00                 77.00         0.00
```

Patients' lab results when blood was taken was stored in the table 'bloodculture', this table contained data on when blood was taken, which bacteria the blood was tested for and whether a positive result was returned or not. Extraction of this data to merge with the master table proved difficult as data was captured as multiple tests for the same patient, with different organisms, resulting in multiple rows for each patient. As such the decision was made to keep just the flag variable as to whether a patient tested positive for any bacteria. This dataset was left joined to the master table on hour and ICU ID, missing values were filled in with a zero value, as either no test was conducted, or no positive result recorded for the patient.

[130]:
```
# load in blood culture dataset
blood_culture = pd.read_csv('mimic_data/bloodculture.csv')
# keeping relevent rows
blood_culture = blood_culture[['icustay_id', 'dy', 'hr', 'positiveculture']]
blood_culture.head()
```

[130]:
```
   icustay_id   dy    hr  positiveculture
0   217870.00 4.00 71.00                1
1   217870.00 4.00 71.00                1
2   217870.00 4.00 71.00                1
3   217870.00 4.00 71.00                1
4   217870.00 5.00 93.00                0
```

[131]:
```
# merge blood culture with master
master = pd.merge(master, blood_culture, how='left')
# replace positive culture NAN values with 0 as no bacteria were tested for
master['positiveculture'] = master['positiveculture'].fillna(0)
master.head()
```

[131]:
```
   icustay_id  hadm_id  subject_id              intime             outtime  \
0      200001   152234       55973 2181-11-25 19:06:12 2181-11-28 20:59:25
1      200001   152234       55973 2181-11-25 19:06:12 2181-11-28 20:59:25
2      200001   152234       55973 2181-11-25 19:06:12 2181-11-28 20:59:25
3      200001   152234       55973 2181-11-25 19:06:12 2181-11-28 20:59:25
4      200001   152234       55973 2181-11-25 19:06:12 2181-11-28 20:59:25

            starttime             endtime     hr   dy gender  \
0 2181-11-24 19:06:12 2181-11-24 20:06:12 -24.00 0.00      F
1 2181-11-24 20:06:12 2181-11-24 21:06:12 -23.00 0.00      F
2 2181-11-24 21:06:12 2181-11-24 22:06:12 -22.00 0.00      F
3 2181-11-24 22:06:12 2181-11-24 23:06:12 -21.00 0.00      F
4 2181-11-24 23:06:12 2181-11-25 00:06:12 -20.00 0.00      F

                  dob  age_years  los  icu_expire_flag   gcs  endotrachflag  \
0 2120-10-31 00:00:00      61.00 3.08                0 13.00           0.00
1 2120-10-31 00:00:00      61.00 3.08                0 13.00           0.00
2 2120-10-31 00:00:00      61.00 3.08                0 13.00           0.00
3 2120-10-31 00:00:00      61.00 3.08                0 13.00           0.00
4 2120-10-31 00:00:00      61.00 3.08                0 13.00           0.00

   spo2  heartrate  meanarterialpressure  urineoutput  positiveculture
0 98.00      92.00                 77.00         0.00             0.00
1 98.00      92.00                 77.00         0.00             0.00
2 98.00      92.00                 77.00         0.00             0.00
3 98.00      92.00                 77.00         0.00             0.00
4 98.00      92.00                 77.00         0.00             0.00
```

Administering of an antibiotic in the ICU was seen as an important variable to capture as this could infer the presence of infection in a patient. Data captured on antibiotics administered to patients was stored in the ‘antibiotics’ table, where data on which antibiotic, the rate given per hour and total amount administered were captured. This data required significant cleaning and modifying to be included, as such the decision was made to create an antibiotic flag variable in the master table based off if a patients ICU ID was present in the ‘antibiotics’ table.

[132]:
```
# load in antibiotics dataset
antibiotics = pd.read_csv('mimic_data/antibiotics.csv')
antibiotics.head()
```

[132]:
```
   icustay_id           starttime             endtime  amount amountuom  \
0   200033.00 2198-08-11 14:40:00 2198-08-11 14:41:00    1.00      dose
```

```
1   200033.00  2198-08-11 15:36:00  2198-08-11 15:37:00    1.00      dose
2   200033.00  2198-08-11 22:05:00  2198-08-11 22:06:00    1.00      dose
3   200033.00  2198-08-12 00:24:00  2198-08-12 00:25:00    1.00      dose
4   200033.00  2198-08-12 00:25:00  2198-08-12 00:26:00    1.00      dose

   rate  rateuom    ordercategoryname  patientweight  totalamount  \
0   NaN      NaN  08-Antibiotics (IV)          74.00       100.00
1   NaN      NaN  08-Antibiotics (IV)          74.00       200.00
2   NaN      NaN  08-Antibiotics (IV)          74.00       100.00
3   NaN      NaN  08-Antibiotics (IV)          74.00       200.00
4   NaN      NaN  08-Antibiotics (IV)          74.00       200.00

  totalamountuom statusdescription                            label  \
0             ml   FinishedRunning  Piperacillin/Tazobactam (Zosyn)
1             ml   FinishedRunning                    Ciprofloxacin
2             ml   FinishedRunning                     Piperacillin
3             ml         Rewritten                    Ciprofloxacin
4             ml   FinishedRunning                    Ciprofloxacin

                      abbreviation  antibiotic    dbsource
0  Piperacillin/Tazobactam (Zosyn)           1  metavision
1                    Ciprofloxacin           1  metavision
2                     Piperacillin           1  metavision
3                    Ciprofloxacin           1  metavision
4                    Ciprofloxacin           1  metavision
```

[133]:

```
# create flag variable if antibiotics given to patient
master['antibiotics_flag'] = np.where(master['icustay_id'].
 ↪isin(antibiotics['icustay_id']),1,0)
# Sanity check
#master['antibiotics_flag'].value_counts()
```

MIMIC-II captures a significant amount of data on mechanical ventilation, including measurements on airway pressures, inspiration and expiration lengths and volume. This dataset contained large amounts of missing/ blank data and selection of variables would be required to accurately capture mechanical ventilation in the models. As such the decision was made to create a flag variable in the master table as to whether a patient was at any point on mechanical ventilation whilst in ICU.

[134]:

```
# load in pv_mechvent dataset
mechvent = pd.read_csv('mimic_data/pv_mechvent.csv')
mechvent.head()
```

[134]:

```
   icustay_id            charttime            starttime              endtime  \
0      200003  2199-08-03 18:00:00  2199-08-03 18:00:00  2199-08-07 13:00:00
1      200003  2199-08-03 19:00:00  2199-08-03 18:00:00  2199-08-07 13:00:00
2      200003  2199-08-03 23:00:00  2199-08-03 18:00:00  2199-08-07 13:00:00
3      200003  2199-08-04 03:00:00  2199-08-03 18:00:00  2199-08-07 13:00:00
```

```
4      200003  2199-08-04 04:00:00  2199-08-03 18:00:00  2199-08-07 13:00:00

   duration_hours  ventnum  minutevolume  settidalvolume  obstidalvolume  \
0           91.00     1.00         12.60          600.00          582.00
1           91.00     1.00         11.20          600.00          610.00
2           91.00     1.00         10.40          600.00          574.00
3           91.00     1.00          9.80          600.00          613.00
4           91.00     1.00         18.00          600.00          575.00

   sponttidalvolume  setpeep  totalpeep  pressurehighaprv  pressurelowaprv  \
0               NaN     5.00        NaN               NaN              NaN
1               NaN    10.00        NaN               NaN              NaN
2               NaN    10.00        NaN               NaN              NaN
3               NaN    10.00        NaN               NaN              NaN
4               NaN     8.00        NaN               NaN              NaN

   timehighaprv  timelowaprv  meanairwaypressure  peakinsppressure  \
0           NaN          NaN               12.00             26.00
1           NaN          NaN               15.00             26.00
2           NaN          NaN               15.00             26.00
3           NaN          NaN               15.00             27.00
4           NaN          NaN               16.00             28.00

   neginspforce  insptime  plateaupressure
0           NaN       NaN            19.00
1           NaN       NaN            22.00
2           NaN       NaN            24.00
3           NaN       NaN            22.00
4           NaN       NaN              NaN
```

[135]:

```
# create flag variable if patient on mechanical ventilation
master['mechvent_flag'] = np.where(master['icustay_id'].
 ↪isin(mechvent['icustay_id']),1,0)
# Sanity check
#master['mechvent_flag'].value_counts()
```

Administering of an antihypertensive such as a vasopressor is used to raise very low blood pressures and is an indicator of cardiac organ failure. As such inclusion of such a variable can infer on the patient's condition and could greatly improve model performance. This data was stored in the 'vasopressors' table, however much like previous tables, it contained missing values and large amounts of data that was outside the scope of the present study. As such another flag variable was created for the master table as to whether a patients ICU ID appeared in the 'vasopressors' table.

[136]:

```
# load in vasopressor dataset
vasopressors = pd.read_csv('mimic_data/vasopressors.csv')
vasopressors.head()
```

[136]:
```
   icustay_id            starttime              endtime  norepinephrine_rate  \
0   200024.00  2127-03-03 16:15:00  2127-03-03 16:45:00                 0.30
1   200024.00  2127-03-03 16:17:00  2127-03-03 20:35:00                  NaN
2   200024.00  2127-03-03 16:45:00  2127-03-03 17:15:00                 0.20
3   200024.00  2127-03-03 17:15:00  2127-03-03 20:30:00                 0.50
4   200028.00  2133-10-29 17:49:00  2133-10-29 18:11:00                 0.06

   norepinephrine_amount  epinephrine_rate  epinephrine_amount  dopamine_rate  \
0                   0.64               NaN                 NaN            NaN
1                    NaN               NaN                 NaN          20.03
2                   0.43               NaN                 NaN            NaN
3                   6.94               NaN                 NaN            NaN
4                   0.11               NaN                 NaN            NaN

   dopamine_amount  dobutamine_rate  dobutamine_amount
0              NaN              NaN                NaN
1           365.96              NaN                NaN
2              NaN              NaN                NaN
3              NaN              NaN                NaN
4              NaN              NaN                NaN
```

[137]:
```
# create flag variable if patient was given vasopressors --> indicates chronic␣
↪organ failure (attempting to raise blood pressure)
master['vasopressor_flag'] = np.where(master['icustay_id'].
↪isin(vasopressors['icustay_id']),1,0)
# Sanity check
#master['vasopressor_flag'].value_counts()
```

[138]:
```
# drop uneeded columns
master = master.drop(['starttime', 'endtime'], axis=1)
master.head()
```

[138]:
```
   icustay_id  hadm_id  subject_id              intime             outtime  \
0      200001   152234       55973 2181-11-25 19:06:12 2181-11-28 20:59:25
1      200001   152234       55973 2181-11-25 19:06:12 2181-11-28 20:59:25
2      200001   152234       55973 2181-11-25 19:06:12 2181-11-28 20:59:25
3      200001   152234       55973 2181-11-25 19:06:12 2181-11-28 20:59:25
4      200001   152234       55973 2181-11-25 19:06:12 2181-11-28 20:59:25

      hr   dy gender                  dob  age_years  los  icu_expire_flag  \
0 -24.00 0.00      F  2120-10-31 00:00:00      61.00 3.08                0
1 -23.00 0.00      F  2120-10-31 00:00:00      61.00 3.08                0
2 -22.00 0.00      F  2120-10-31 00:00:00      61.00 3.08                0
3 -21.00 0.00      F  2120-10-31 00:00:00      61.00 3.08                0
4 -20.00 0.00      F  2120-10-31 00:00:00      61.00 3.08                0

    gcs  endotrachflag  spo2  heartrate  meanarterialpressure  urineoutput  \
```

```
0 13.00            0.00 98.00      92.00                      77.00         0.00
1 13.00            0.00 98.00      92.00                      77.00         0.00
2 13.00            0.00 98.00      92.00                      77.00         0.00
3 13.00            0.00 98.00      92.00                      77.00         0.00
4 13.00            0.00 98.00      92.00                      77.00         0.00

   positiveculture  antibiotics_flag  mechvent_flag  vasopressor_flag
0             0.00                 0              0                 0
1             0.00                 0              0                 0
2             0.00                 0              0                 0
3             0.00                 0              0                 0
4             0.00                 0              0                 0
```

[139]:
```
# memory management to optimise RAM
del pt_stay_hr
del patients
del pt_icu_outcome
del gcs_hourly
del vitals_hourly
del output_hourly
del blood_culture
del antibiotics
del mechvent
del vasopressors
```

2.2 Further cleaning

[140]:
```
# Define function to examine percentage of missing data by column
def missing_data(df):
    for column in df.columns:
        print(f'Column {column}', f'has {100 * sum(df[column].isnull())/len(df):
↪.2f}% missing data')
        print()
    return
```

[141]:
```
missing_data(master)
```

```
Column icustay_id has 0.00% missing data

Column hadm_id has 0.00% missing data

Column subject_id has 0.00% missing data

Column intime has 0.00% missing data

Column outtime has 0.00% missing data
```

```
Column hr has 0.00% missing data

Column dy has 0.04% missing data

Column gender has 0.00% missing data

Column dob has 0.00% missing data

Column age_years has 0.00% missing data

Column los has 0.00% missing data

Column icu_expire_flag has 0.00% missing data

Column gcs has 0.00% missing data

Column endotrachflag has 0.00% missing data

Column spo2 has 0.00% missing data

Column heartrate has 0.00% missing data

Column meanarterialpressure has 0.00% missing data

Column urineoutput has 0.00% missing data

Column positiveculture has 0.00% missing data

Column antibiotics_flag has 0.00% missing data

Column mechvent_flag has 0.00% missing data

Column vasopressor_flag has 0.00% missing data
```

[142]:

```
# Examine the master dataframe
master.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 3735155 entries, 0 to 3735154
Data columns (total 22 columns):
 #   Column                Dtype
---  ------                -----
 0   icustay_id            int64
 1   hadm_id               int64
 2   subject_id            int64
 3   intime                object
 4   outtime               object
```

```
 5   hr                    float64
 6   dy                    float64
 7   gender                object
 8   dob                   object
 9   age_years             float64
 10  los                   float64
 11  icu_expire_flag       int64
 12  gcs                   float64
 13  endotrachflag         float64
 14  spo2                  float64
 15  heartrate             float64
 16  meanarterialpressure  float64
 17  urineoutput           float64
 18  positiveculture       float64
 19  antibiotics_flag      int64
 20  mechvent_flag         int64
 21  vasopressor_flag      int64
dtypes: float64(11), int64(7), object(4)
memory usage: 655.4+ MB
```

With the final master table compiled, time variables were altered to encode as pandas datetime objects so they could be read correctly. The variable 'gender' was encoded to a binary variable whereby 1 represented male and 0 representing female.

[143]:
```
# convert intime, outtime, dob to datetime format
# Note: dates/ times have been shifted to preserve patient identity as per␣
↪https://mimic.physionet.org/mimicdata/time/ hence they are set into the next␣
↪century
master['intime'] = pd.to_datetime(master['intime'])
master['outtime'] = pd.to_datetime(master['outtime'])
master['dob'] = pd.to_datetime(master['dob'])
```

[144]:
```
# encode gender to binary variable
master['gender'] = np.where(master['gender']=='M',1,0)
```

Examination of the distribution of continuous variables showed values that appeared outside both normal and abnormal clinical ranges. SpO2 (blood oxygen concentration) was cleaned to ensure values between 0-100, Heart rate was cleaned to only contain values from 0-250bpm as this was deemed the maximum heart rate from clinical sources, MAP was cleaned to contain data from 0-150 and urine output was set from 0-500ml which was deemed the maximum storage capacity of the bladder from clinical sources. The dataset was then copied to be used for the statistical model before being altered for the machine learning model.

[145]:
```
# Examine the range of values in dataframe
master.describe()
```

[145]:
```
       icustay_id    hadm_id  subject_id         hr         dy     gender  \
count  3735155.00 3735155.00  3735155.00 3735154.00 3733756.00 3735155.00
```

```
mean    221006.17  149911.41    29813.51     233.70      10.28         0.55
std      12145.79   28783.27    26362.53     422.45      17.60         0.50
min     200001.00  100003.00        3.00     -24.00       0.00         0.00
25%     210579.00  125146.00    10534.00      11.00       1.00         0.00
50%     220971.00  149733.00    21280.00      68.00       3.00         1.00
75%     231549.00  174744.00    41446.00     256.00      11.00         1.00
max     241864.00  199999.00    99995.00    4067.00     170.00         1.00

       age_years        los  icu_expire_flag        gcs  endotrachflag  \
count 3735155.00 3735155.00       3735155.00 3735155.00     3735155.00
mean       48.60      20.49             0.01      12.74           0.08
std        30.81      28.53             0.08       1.66           0.26
min         0.00       0.00             0.00       3.00           0.00
25%        23.00       2.90             0.00      13.00           0.00
50%        58.00       8.57             0.00      13.00           0.00
75%        74.00      24.81             0.00      13.00           0.00
max        91.40     169.42             1.00      15.00           1.00

            spo2  heartrate  meanarterialpressure  urineoutput  \
count 3735155.00 3735155.00            3735155.00   3735155.00
mean       96.80      99.74                 78.25        48.61
std      3751.41      53.22                 94.51      2371.05
min         0.00       0.00                -25.00     -4400.00
25%        97.00      83.00                 76.00         0.00
50%        98.00      92.00                 77.00         0.00
75%        98.50     105.00                 78.00        50.00
max   6363333.00   86101.00             120130.03   4555555.00

       positiveculture  antibiotics_flag  mechvent_flag  vasopressor_flag
count       3735155.00        3735155.00     3735155.00        3735155.00
mean              0.01              0.23           0.67              0.23
std               0.11              0.42           0.47              0.42
min               0.00              0.00           0.00              0.00
25%               0.00              0.00           0.00              0.00
50%               0.00              0.00           1.00              0.00
75%               0.00              0.00           1.00              0.00
max               1.00              1.00           1.00              1.00
```

[146]:
```
# clean up spo2 column - cannot have an O2 saturation over 100
# spo2 (range allowed 0-100)
master = master[master['spo2']<=100]


# heartrate (max HR is about 200 bpm --> hence make range allowed 0-250
# https://www.heart.org/en/healthy-living/fitness/fitness-basics/
 ↪target-heart-rates
# clean up heart rate column - range 0-250bpm
```

```
master = master.loc[(master['heartrate']>= 0) & (master['heartrate']<= 250)]



# meanarterialpressure ( average range is 60-110mmHg, hence for the purposes␣
↪here we will make range 0-150
# https://en.wikipedia.org/wiki/Mean_arterial_pressure )
# clean up MAP column - range 0-150
master = master.loc[(master['meanarterialpressure']>= 0) &␣
↪(master['meanarterialpressure']<= 150)]


# urineoutput (cannot be negative, normal maximum capacity is 300-400ml, hence␣
↪make range 0-500ml
# https://www.ncbi.nlm.nih.gov/pmc/articles/PMC3206217/
# clean up urine output - range 0-500
master = master.loc[(master['urineoutput']>= 0) & (master['urineoutput']<= 500)]
```

[147]:
```
# Sanity check
#master.describe()
```

[148]:
```
# Copy dataset for task 2
task2 = master
```

2.3 Task 1

Data was segmented to contain only the first 24 hours of ICU patient recorded data, unrequired patient identifiers were dropped and encoding of continuous variables were corrected. The entire master table was grouped by each patient's ICU ID and the mean values taken. This condensed all hourly values into one value for each variable as to simplify the model and reduce training time. Mean values were taken to ensure outliers/ abnormal values were somewhat captured.

[149]:
```
# Segment master dataset for data from first 24 hours in ICU (ie where hr␣
↪column = 0-24) for Task 1
master = master.loc[(master['hr']>= 0) & (master['hr']<= 24)]
```

[150]:
```
# Drop unwanted rows for machine learning
master = master.drop(['hadm_id', 'subject_id', 'hr', 'dy'], axis=1)
```

[151]:
```
master['positiveculture'] = master['positiveculture'].astype(int)
master['endotrachflag'] = master['endotrachflag'].astype(int)
```

[152]:
```
# Sanity Check
#master.info()
```

```
[153]: # Group and average all values by each icustay_id - effectively reduces all␣
        ↪measurements down to an average across each patients icu stay
       master = master.groupby('icustay_id').mean().reset_index()
```

```
[154]: # Sanity Check
       #master.head()
```

2.3.1 Machine Learning

```
[155]: # check the number of patients who died in the ICU and the balance/ imblance of␣
        ↪the dataset
       master['icu_expire_flag'].value_counts()
```

```
[155]: 0.00    25661
       1.00      125
       Name: icu_expire_flag, dtype: int64
```

A very imbalanced dataset. Roughly ~0.5% of patients died in ICU. This will lead our models to always predict that the patient will live out their ICU stay. Here we will use the imbalanced-learn library to use a common technique called SMOTE (Synthetic Minority Oversampling Technique), whereby synthetic samples from the minority class are generated to aid model performance.

The dataset was then split into training and testing splits (80%/20%) and stratified to ensure even number of mortality cases in both the training and testing splits.

```
[156]: # Split dataset into X and Y variables
       y = master['icu_expire_flag']
       X = master.drop(axis=1, columns=['icu_expire_flag','icustay_id'])

       # Using SMOTE to balance data and create 20% mortality
       smote = SMOTE(sampling_strategy=0.2,random_state=0)
       X_resample, y_resample = smote.fit_resample(X,y)

       # Train test split - 80% train, 20% test
       # Set random state for reproducability and stratify data to ensure same␣
        ↪proportion of Y variables in each split
       X_train, X_test, y_train, y_test = train_test_split(X_resample, y_resample,␣
        ↪test_size=0.2, random_state=0, stratify=y_resample)
```

```
[157]: # Sanity Check
       y_resample.value_counts()
```

```
[157]: 0.00    25661
       1.00     5132
       Name: icu_expire_flag, dtype: int64
```

2.3.2 Logistic Regression model

A logistic regression model was defined as well as a scalar to standardise features by removing the mean and scaling to the unit variance. A pipeline was defined before being passed to a hyperparameter grid to find the models optimal parameters.

[158]:
```
# Set seed
np.random.seed(1111)
# Define model
log_reg = LogisticRegression(max_iter=1000)

# Scale features to make training easier using Standard Scalar
standard_scalar = StandardScaler()

# Using a pipline to scale and apply to model
pipe_logistic = Pipeline([('Transform', standard_scalar),('Estimator',␣
↪log_reg)])

# Define a hyperparameter grid
parameter_grid_logistic = {'Estimator__C':[0.001, 0.01, 0.1, 1, 10, 100],
                           'Estimator__penalty':['l1','l2'],
                           'Estimator__class_weight':['None', 'balanced',{0:0.
↪9, 1:0.1}, {0:0.95,1:0.05}, {0:0.85,1:0.15},{0:0.80,1:0.20}],
                           'Estimator__solver':
↪['liblinear','lbfgs','newton-cg','sag','saga']}
```

A GridSearch function was passed to the pipeline to perform 5-fold cross validation on the dataset, serially searching each hyperparameter set to find optimal. Scoring was conducted using an F1 score, which is calculated from the model's precision and recall which came to 0.88.

[159]:
```
# Assign Grid Search CV to the pipeline and hyperparameter grid
grid_search_logistic = GridSearchCV(pipe_logistic, parameter_grid_logistic,␣
↪cv=5, scoring='f1_weighted', n_jobs=-1)

# Fit the grid search and pipeline
grid_search_logistic.fit(X_train, y_train)

# Print the best parameters found and best score found
print("Best hyper-parameters for Logistic Regression: {}".
↪format(grid_search_logistic.best_params_))
print("Best cross-validation average f1-score for Logistic Regression: {:.3f}".
↪format(grid_search_logistic.best_score_))
```

```
/Users/joshbryden/opt/miniconda3/envs/capstone/lib/python3.7/site-
packages/sklearn/model_selection/_search.py:921: UserWarning: One or more of the
test scores are non-finite: [       nan        nan        nan        nan
0.78474481        nan
 0.8446286  0.8446613  0.8446613  0.8446613  0.83491118        nan
```

```
       nan        nan 0.86405162 0.84322166 0.86101178 0.86101178
0.86101178 0.86101178 0.75755675        nan        nan        nan
0.75755675 0.75753662 0.75755675 0.75755675 0.75755675 0.75755675
0.75755675        nan        nan        nan 0.75755675 0.75755675
0.75755675 0.75755675 0.75755675 0.75755675 0.75755675        nan
       nan        nan 0.75755675 0.75761435 0.75755675 0.75755675
0.75755675 0.75755675 0.75755675        nan        nan        nan
0.75755675 0.7606286  0.75755675 0.75755675 0.75755675 0.75755675
       nan        nan        nan        nan 0.87402515        nan
0.8762138  0.8762138  0.87616247 0.87616247 0.87323736        nan
       nan        nan 0.87500482 0.8713032  0.87423087 0.87423087
0.87423087 0.87429922 0.75753662        nan        nan        nan
0.75753662 0.75753662 0.75749635 0.75749635 0.75749635 0.75749635
0.75755675        nan        nan        nan 0.75755675 0.75755675
0.75755675 0.75755675 0.75755675 0.75755675 0.75975231        nan
       nan        nan 0.76067683 0.75879078 0.75900041 0.75900041
0.75900041 0.75900041 0.76267031        nan        nan        nan
0.76279578 0.76911497 0.76974081 0.76974081 0.76974081 0.76974081
       nan        nan        nan        nan 0.88072341        nan
0.88036399 0.88036399 0.88036399 0.88036399 0.8759382         nan
       nan        nan 0.87617131 0.87549952 0.87613044 0.87613044
0.87613044 0.87613044 0.75832155        nan        nan        nan
0.75839793 0.75861225 0.75882145 0.75882145 0.75882145 0.75882145
0.75753662        nan        nan        nan 0.75753662 0.75751648
0.75751648 0.75751648 0.75751648 0.75751648 0.77304904        nan
       nan        nan 0.77365162 0.77313808 0.77482889 0.77482889
0.77482889 0.77482889 0.7932071         nan        nan        nan
0.79372137 0.79361452 0.79565216 0.79565216 0.79565216 0.79565216
       nan        nan        nan        nan 0.88118081        nan
0.8812288  0.8812288  0.8812288  0.8812288  0.87644878        nan
       nan        nan 0.87644878 0.87652362 0.87648307 0.87648307
0.87648307 0.87648307 0.76056105        nan        nan        nan
0.76056105 0.760465   0.76082709 0.76082709 0.76082709 0.76082709
0.75745609        nan        nan        nan 0.75745609 0.75745609
0.75745609 0.75745609 0.75745609 0.75745609 0.78093161        nan
       nan        nan 0.78093161 0.78051941 0.78099467 0.78099467
0.78099467 0.78099467 0.80009128        nan        nan        nan
0.8002193  0.79981412 0.80024623 0.80024623 0.80024623 0.80024623
       nan        nan        nan        nan 0.88145261        nan
0.88145261 0.88145261 0.88145261 0.88145261 0.87656452        nan
       nan        nan 0.87659876 0.87652397 0.87655822 0.8765174
0.8765174  0.8765174  0.76090157        nan        nan        nan
0.76090157 0.76090157 0.76090157 0.76090157 0.76090157 0.76090157
0.75745609        nan        nan        nan 0.75745609 0.75745609
0.75745609 0.75745609 0.75745609 0.75745609 0.78129376        nan
       nan        nan 0.78129376 0.78129376 0.78126841 0.78126841
0.78126841 0.78126841 0.80055241        nan        nan        nan
0.80055241 0.80055241 0.80055241 0.80055241 0.80055241 0.80055241
```

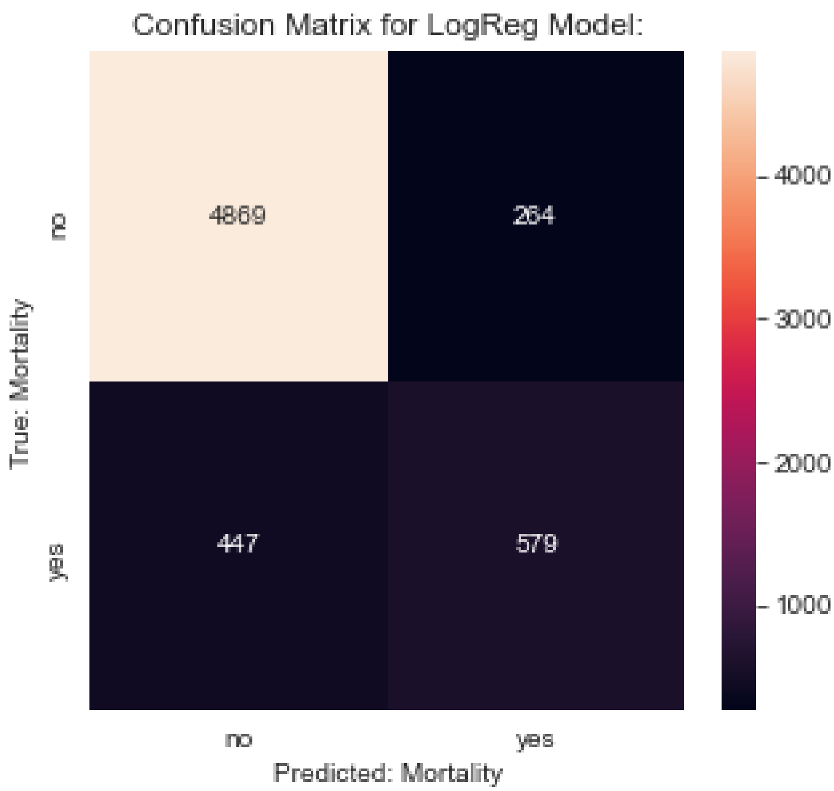
```
        nan        nan        nan        nan 0.88145261        nan
 0.88145261 0.88145261 0.88145261 0.88145261 0.87656452        nan
        nan        nan 0.87659876 0.87659876 0.87659876 0.87659876
 0.87659876 0.87659876 0.76090157        nan        nan        nan
 0.76090157 0.76090157 0.76090157 0.76090157 0.76090157 0.76090157
 0.75745609        nan        nan        nan 0.75745609 0.75745609
 0.75745609 0.75745609 0.75745609 0.75745609 0.78124384        nan
        nan        nan 0.78124384 0.78124384 0.78124384 0.78124384
 0.78124384 0.78124384 0.80075636        nan        nan        nan
 0.80075636 0.80075636 0.80075636 0.80075636 0.80075636 0.80075636]
  category=UserWarning
Best hyper-parameters for Logistic Regression: {'Estimator__C': 10,
'Estimator__class_weight': 'None', 'Estimator__penalty': 'l1',
'Estimator__solver': 'saga'}
Best cross-validation average f1-score for Logistic Regression: 0.881
```

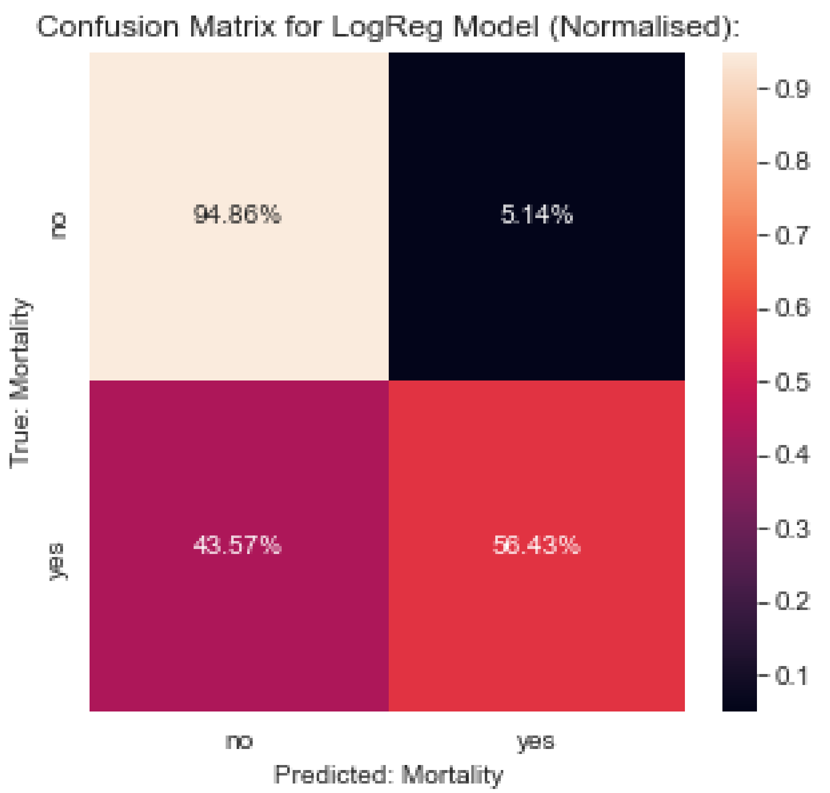
Using the generated model, predictions on the test data were completed and a confusion matrix generated, showing the True values and model predicted values. The true positive rate of the model was 56.43% and the false positive rate of 5.14%. This means that the model, whilst not perfect, captures 56.43% of all mortality in the ICU and only incorrectly classifies mortality 5.14% of the time.

[160]:
```python
# Predict on training
y_pred_training = grid_search_logistic.predict(X_train)

# Predict on test
y_pred_test = grid_search_logistic.predict(X_test)
```

[161]:
```python
# Create confusion matrix to examine where and what the model is predicting
confusion_logreg = confusion_matrix(y_test, y_pred_test)
confusion_logreg_normalised = confusion_matrix(y_test, y_pred_test,␣
 ↪normalize='true')
figs, axes = plt.subplots(2, 1, figsize=(5, 10))
sns.heatmap(confusion_logreg, annot=True, ax=axes[0], fmt='.0f')
sns.heatmap(confusion_logreg_normalised, annot=True, ax=axes[1], fmt='.2%')

# Labels
for i in (0, 1):
    axes[i].set_xlabel('Predicted: Mortality')
    axes[i].set_ylabel('True: Mortality')
    axes[i].xaxis.set_ticklabels(['no','yes'])
    axes[i].yaxis.set_ticklabels(['no','yes'])
axes[0].set_title('Confusion Matrix for LogReg Model:')
axes[1].set_title('Confusion Matrix for LogReg Model (Normalised):')
```

[161]:
```
Text(0.5, 1.0, 'Confusion Matrix for LogReg Model (Normalised):')
```


Confusion Matrix for LogReg Model:
4869
264
447
579
True: Mortality
no
yes
Predicted: Mortality
4000
3000
2000
1000


Confusion Matrix for LogReg Model (Normalised):
94.86%
5.14%
43.57%
56.43%
True: Mortality
no
yes
Predicted: Mortality
0.9
0.8
0.7
0.6
0.5
0.4
0.3
0.2
0.1

To understand the model in depth, the Lime package was used to examine which features the model favours in its prediction. Examining the 20,000th patient we can see which features led to the decision to predict that the patient will live. The logistic regression model favours the antibiotics flag, GCS, vasopressor flag and endotrach flag the highest for predicting mortality inside the ICU. This is not surprising as these variables capture a majority of the sources of disease or complications that could lead to death. Presence of infection (and/or sepsis), levels of consciousness/ brain function, cardiac function and respiratory system function are all captured in this model.

[162]:
```
# Using LIME package to explain model components
explainer = lime.lime_tabular.LimeTabularExplainer(X_train.values,
                                                    feature_names=X_train.
 ↪columns.values.tolist(),
                                                    class_names=np.
 ↪unique(y_train))

# Print out the 20,000th row
master.loc[[20000]]
```

[162]:
```
       icustay_id  gender  age_years  los  icu_expire_flag    gcs  \
20000      232447    1.00      72.00 4.68             0.00  11.00

       endotrachflag  spo2  heartrate  meanarterialpressure  urineoutput  \
20000           0.35 98.16      91.86                 83.19        63.27

       positiveculture  antibiotics_flag  mechvent_flag  vasopressor_flag
20000             0.00              0.00           1.00              0.00
```

[163]:
```
exp = explainer.explain_instance(X_train.values[20000], grid_search_logistic.
 ↪predict_proba, num_features=14)
exp.show_in_notebook(show_all=False, show_table=True)
```

```
<IPython.core.display.HTML object>
```

2.3.3 Random Forest

A random forest model was also applied to the dataset to see if this had any effect over the logistic regression model. The model was defined, and scaling was not required as the random forest stems from Tree based model whereby specific features values are used (as opposed to scaled values). A hyperparameter grid was defined and initially a RandomSearchCV was called upon to find optimal parameters without a serial search due to training time constraints. Fivefold cross validation was used and scored with an F1 score. After optimal parameters were found, the hyperparameter grid was refined to include a range of values for each parameter that were above and below the 'optimal' parameters found by RandomSearchCV. This new hypermeter grid was passed to GridSearchCV and with fivefold cross validation and the random forest optimal model parameters found.

[164]:
```
# Set seed
np.random.seed(1111)
```

```
# Define our Random Forest
RF = RandomForestClassifier(n_jobs=-1, random_state=0, min_samples_leaf=5)

# No Scaling required

# Define hyperparameter grid for Random Search CV to narrow down hyperparameters
random_forest_parameter_grid = {'n_estimators': [10, 50, 100, 200, 500, 1000,␣
↪1200, 1300, 1500],
                    'min_samples_split': [int(x) for x in np.linspace(start=2,␣
↪stop=15)],
                    'min_samples_leaf': [int(x) for x in np.linspace(start=2,␣
↪stop=15)],
                    'max_depth': [int(x) for x in np.linspace(start=2,␣
↪stop=30)],
                    'max_features': ['auto','sqrt','log2']}
```

[165]:
```
# Assign Randomsearch CV to our parameter grid
random_search_random_forest = RandomizedSearchCV(RF,␣
↪random_forest_parameter_grid, cv=5, scoring='f1_weighted', n_jobs=-1)

# Fit model
random_search_random_forest.fit(X_train, y_train)

# Print the best parameters found and best score found
print("Best hyper-parameters for Random Forest: {}".
↪format(random_search_random_forest.best_params_))
print("Best cross-validation average f1-score for Random Forest: {:.3f}".
↪format(random_search_random_forest.best_score_))
```

```
Best hyper-parameters for Random Forest: {'n_estimators': 500,
'min_samples_split': 12, 'min_samples_leaf': 3, 'max_features': 'sqrt',
'max_depth': 29}
Best cross-validation average f1-score for Random Forest: 0.991
```

[166]:
```
# After narrowing down the hyperparameters perform a GridSearchCV
grid_parameter_search = {'n_estimators': [int(x) for x in np.
↪linspace(start=200, stop=1000, num=5)],
                    'min_samples_split': [int(x) for x in np.linspace(start=8,␣
↪stop=16, num=5)],
                    'min_samples_leaf': [int(x) for x in np.linspace(start=2,␣
↪stop=10, num=5)],
                    'max_depth': [int(x) for x in np.linspace(start=5, stop=15,␣
↪num=5)],
                    'max_features': ['sqrt']}
```

```
# GridSearchCV
grid_search_RF = GridSearchCV(RF, grid_parameter_search, cv=5,␣
↪scoring='f1_weighted', n_jobs=-1)

# Fit model
grid_search_RF.fit(X_train, y_train)

# Print the best parameters found and best score found
print("Best hyper-parameters for Random Forest: {}".format(grid_search_RF.
↪best_params_))
print("Best cross-validation average f1-score for Random Forest: {:.3f}".
↪format(grid_search_RF.best_score_))
```

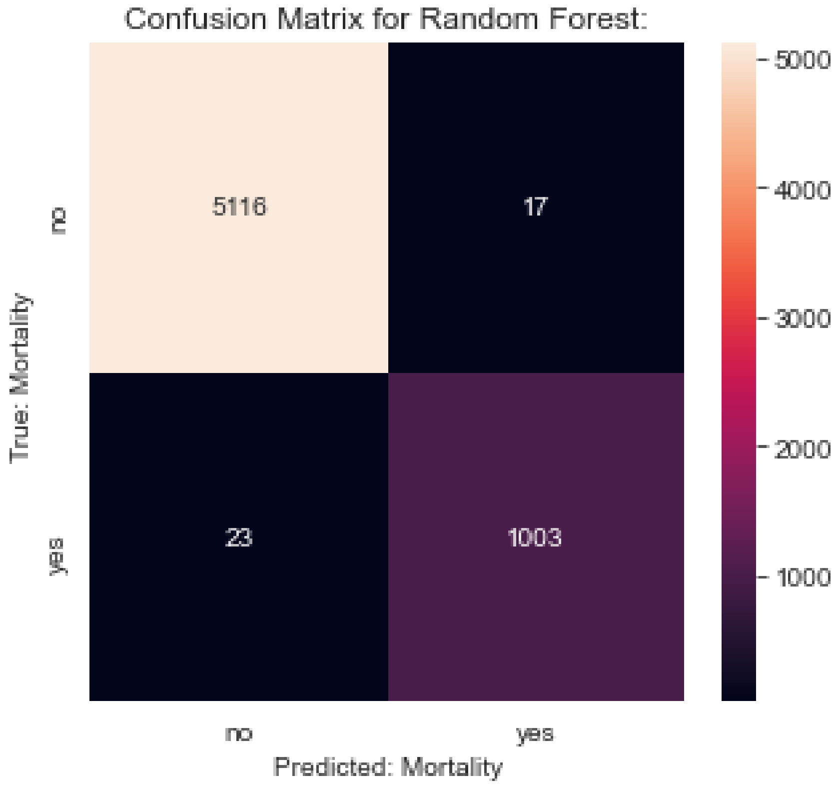
```
Best hyper-parameters for Random Forest: {'max_depth': 15, 'max_features':
'sqrt', 'min_samples_leaf': 2, 'min_samples_split': 8, 'n_estimators': 1000}
Best cross-validation average f1-score for Random Forest: 0.992
```

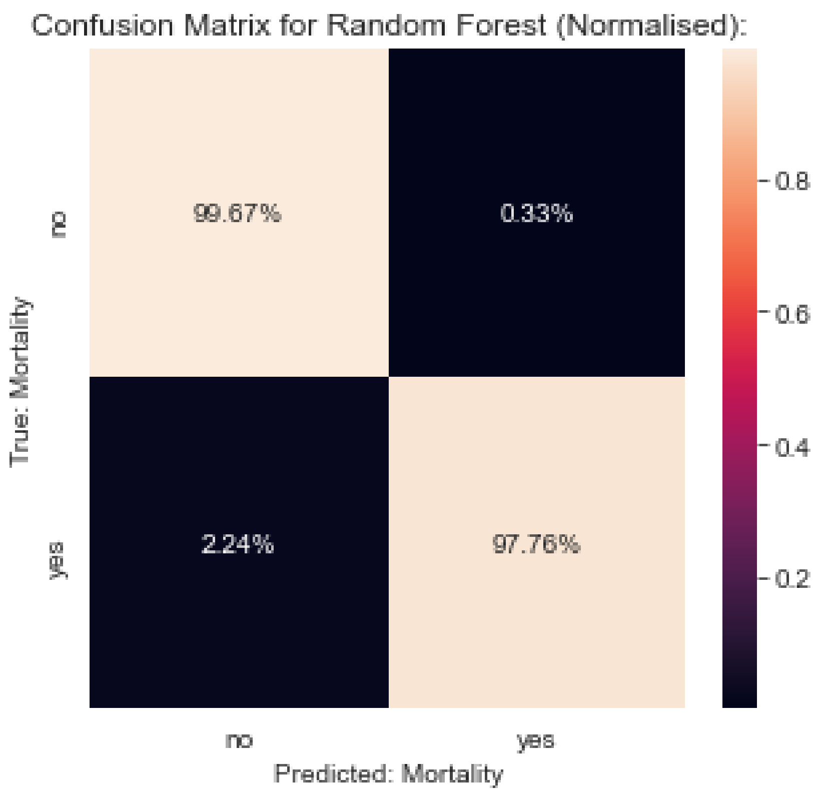
Using this model, predictions were made on the test data and a confusion matrix generated. The matrix shows that the random forest model had a true positive rate of 97.76% and a false positive rate of 0.33%. Hence that the model accurately predicts 97.76% of mortality inside the ICU (1003 cases) and will only incorrectly predict mortality in 0.33% of patients (17 cases). This present a significance improvement over the Logistic Regression model, however the logistic regression model is not exposed to overfitting in the data, whereas tree-based models are.

[167]:
```
# Predict on the training set:
y_pred_RF_training= grid_search_RF.predict(X_train)

# Predict on the test set:
y_pred_RF_test = grid_search_RF.predict(X_test)
```

[168]:
```
# Setting up the confusion matrices:
confusion_RF = confusion_matrix(y_test, y_pred_RF_test)
confusion_RF_normalised = confusion_matrix(y_test, y_pred_RF_test,␣
↪normalize='true')
figs, axes = plt.subplots(2, 1, figsize=(5, 10))
sns.heatmap(confusion_RF, annot=True, ax=axes[0], fmt='.0f')
sns.heatmap(confusion_RF_normalised, annot=True, ax=axes[1], fmt='.2%')

# Labels
for i in (0, 1):
    axes[i].set_xlabel('Predicted: Mortality')
    axes[i].set_ylabel('True: Mortality')
    axes[i].xaxis.set_ticklabels(['no','yes'])
    axes[i].yaxis.set_ticklabels(['no','yes'])
axes[0].set_title('Confusion Matrix for Random Forest:')
axes[1].set_title('Confusion Matrix for Random Forest (Normalised):')
```

[168]: Text(0.5, 1.0, 'Confusion Matrix for Random Forest (Normalised):')





The Lime package was used to examine the importance of model features, similar features that appeared in the logistic regression model were of similar importance in the random forest model. Mechanical ventilation flag was considered one of the most important as was the vasopressor flag that both explain the presence of serious underlying complications. Interestingly the model found that gender was the third most importance predictor when examining the 20,000th patient. Further testing will be required to investigate this (if any) effects of gender on the model. Furthermore, the model found that antibiotics, endotracheal breathing tubes and the GCS were all significantly high in informing the models prediction.

[169]:
```
# Using LIME package to explain model components
explainer = lime.lime_tabular.LimeTabularExplainer(X_train.values,
                                                    feature_names=X_train.
 ↪columns.values.tolist(),
                                                    class_names=np.
 ↪unique(y_train))

# Print out the 20,000th row
master.loc[[20000]]
```

[169]:
```
       icustay_id  gender  age_years  los  icu_expire_flag   gcs  \
20000      232447    1.00      72.00 4.68             0.00 11.00

       endotrachflag  spo2  heartrate  meanarterialpressure  urineoutput  \
20000           0.35 98.16      91.86                 83.19        63.27

       positiveculture  antibiotics_flag  mechvent_flag  vasopressor_flag
20000             0.00              0.00           1.00              0.00
```

[170]:
```
exp = explainer.explain_instance(X_train.values[20000], grid_search_RF.
 ↪predict_proba, num_features=14)
exp.show_in_notebook(show_all=False, show_table=True)
```

```
<IPython.core.display.HTML object>
```

Over-fitting occurs when the tree (model) becomes too specific on training data and then does not perform well on test or other unseen data. Our random forest model whilst tuned to ensure that this occurrence is limited still presents signs of overfitting with a near perfect F1 score. Whilst this may be simply the optimal model, further research into hyperparameter tuning would be required to properly examine the effects of further tuning. Future research should split data into training, validation a test data, whereby validation data can be used to tune model parameters further to investigate any effects of overfitting.

2.3.4 Dense Nerual Network

Unfortunately, due to computational and time constraints with model training (blown out to 3+ hours) the Dense Neural Network was discarded from the study.

[171]:
```
# # Set seed
# np.random.seed(1111)

# # https://machinelearningmastery.com/
↪choose-an-activation-function-for-deep-learning/

# # https://machinelearningmastery.com/
↪grid-search-hyperparameters-deep-learning-models-python-keras/

# # Function to create NN
# def neural_network(DropoutL1):
#     # create model
#     model = tensorflow.keras.Sequential()
#     # 1st layer - input shape 14, relu activation for normal neural net,␣
↪weights sampled from        normal distribution
#     model.add(Dense(14, input_shape=(14,),activation='relu',␣
↪kernel_initializer='uniform'))
#     # dropout term
#     model.add(Dropout(DropoutL1))
#     # 2nd layer - as above re 1st layer
#     model.add(Dense(14, activation='relu', kernel_initializer='uniform'))
#     # output layer - sigmoid as output is binary
#     model.add(Dense(1, activation='sigmoid', kernel_initializer='uniform'))
#     # compile model - tf.keras cannot use f1 score hence use accuracy
#     model.compile(optimizer='adam', loss='binary_crossentropy',␣
↪metrics=['acc'])
#     model.summary()
#     return model
```

[172]:
```
# # Keras classifier required to perform GridSearchCV on our function (NN)
# nn = KerasClassifier(build_fn=neural_network)
```

[173]:
```
# # Scale data using standard scalar and pipeline
# pipeline_nn = Pipeline([('Transform', standard_scalar),('Estimator', nn)])

# # Define parameter grid
# parameter_grid_nn = {'Estimator__epochs':[200],
#                'Estimator__batch_size': [25],
#                'Estimator__DropoutL1':[0.1, 0.2, 0.3, 0.4]}

# # parameter_grid_nn = {'Estimator__epochs':[200, 250, 300, 350, 400],
# #                'Estimator__batch_size': [25, 50, 100, 200],
# #                'Estimator__DropoutL1':[0.1, 0.2, 0.3, 0.4]}

# # Define GridsearchCV
# grid_search_nn = GridSearchCV(estimator=pipeline_nn,␣
↪param_grid=parameter_grid_nn, cv=5, n_jobs=-1)
```

```
# # Fit model
# grid_search_nn.fit(X_train, y_train)

# # Print the best parameters found and best score found
# print("Best hyper-parameters for the Neural Network: {}".
↪format(grid_search_nn.best_params_))
# print("Best cross-validation average accuracy for the Neural Network: {:.3f}".
↪format(grid_search_nn.best_score_))
```

2.4 Task 2

The second research question aimed to address the weekend effect in the ICU from the MIMIC-II dataset. The dataset was copied from the master table before being altered for the machine learning models. In order to investigate ICU admission on a weekend, a new variable was created using the admission time ('intime' column) to encode for each day of the week (0 for Monday and 7 for Sunday). From this variable a weekend flag variable was created based off this weekday number.

[174]:
```
# https://pandas.pydata.org/pandas-docs/stable/reference/api/pandas.
↪DatetimeIndex.weekday.html
# Create weekday column to create weekend admission flag varaible - weekend␣
↪will take value of 5 or 6 in 'weekday' column
task2['weekday'] = task2['intime'].dt.weekday
task2['weekend_flag'] = np.where((task2['weekday']>=5),1,0)
task2 = task2.drop(axis=1, columns=['weekday'])
```

Unrequired columns were dropped, and variables encoded correctly.

[175]:
```
# Dataset preprocessing - as per task 1
task2 = task2.drop(['hadm_id', 'subject_id', 'hr', 'dy'], axis=1)
task2['positiveculture'] = task2['positiveculture'].astype(int)
task2['endotrachflag'] = task2['endotrachflag'].astype(int)
```

The dataset was grouped by each ICU ID which reduced each ICU stay down from hourly rows/measurements to a single average value for each variable.

[176]:
```
# Groupby ICU stay and average out values across ICU stay
task2 = task2.groupby('icustay_id').mean().reset_index()
```

[177]:
```
task2['icu_expire_flag'].value_counts()
```

[177]:
```
0.00    25661
1.00      125
Name: icu_expire_flag, dtype: int64
```

[178]:
```
task2['weekend_flag'].value_counts()
```

[178]:
```
0.00    20105
1.00     5681
Name: weekend_flag, dtype: int64
```

Much like the previous research question, SMOTE was used to ensure that there were sufficient ICU mortality cases in the dataset. Before SMOTE there was approximately ~0.5% mortality cases in the dataset, which is much too small for effective analysis. Using SMOTE we created 20% mortality in the dataset to better inform model selection.

[179]:
```
# Split dataset into X and Y variables
y = task2['icu_expire_flag']
X = task2.drop(axis=1, columns=['icu_expire_flag','icustay_id'])

# Select just weekend flag for model 1
X1 = X[['weekend_flag']]
# add intercept constant - column of 1's
X1 = sm.add_constant(X1)

# Using SMOTE to balance data and create 20% mortality
smote = SMOTE(sampling_strategy=0.2,random_state=0)
X_resample, y_resample = smote.fit_resample(X1,y)
```

[180]:
```
# Sanity check
y_resample.value_counts()
```

[180]:
```
0.00    25661
1.00     5132
Name: icu_expire_flag, dtype: int64
```

Due to the binary nature of our target variable and with little to nil time to event information available for death inside the ICU, a logistic regression statistical model was to be used. This allowed for a binary outcome whilst still retaining performance that we obtained in the previous research question. Initially the first model was fitted with just an intercept term and the weekend flag variable. The odds for a simple predictor model found that the odds for weekend admission mortality was 85% higher.

[181]:
```
logistic1_task2 = sm.Logit(endog=y_resample, exog=X_resample).fit(maxiter=1000)
```

```
Optimization terminated successfully.
         Current function value: 0.450271
         Iterations 6
```

[182]:
```
print(logistic1_task2.summary())
```

```
                           Logit Regression Results
==============================================================================
Dep. Variable:        icu_expire_flag   No. Observations:                30793
Model:                          Logit   Df Residuals:                    30791
```

```
Method:                           MLE   Df Model:                            1
Date:                Sun, 09 May 2021   Pseudo R-squ.:               0.0006251
Time:                        18:11:17   Log-Likelihood:                -13865.
converged:                       True   LL-Null:                       -13874.
Covariance Type:            nonrobust   LLR p-value:                 3.117e-05
================================================================================
                  coef    std err          z      P>|z|      [0.025      0.975]
--------------------------------------------------------------------------------
const          -1.5767      0.017    -92.290      0.000      -1.610      -1.543
weekend_flag   -0.1580      0.038     -4.120      0.000      -0.233      -0.083
================================================================================
```

[183]:

```
print(f'Odd ratio for weekend admission to ICU :{np.exp(-0.1580 )}')
```

```
Odd ratio for weekend admission to ICU :0.8538497819684817
```

To investigate the effects of the other features in the dataset, we fitted a model using all the predictors in the dataset. The model required increasing the maximum iterations to 10,000 to ensure model convergence as lower levels of iterations failed to converge. This model shows warnings of quasi separation whereby sections of the dataset could be separated entirely based off a single/selection of variables and their respective values. This model showed a lower log likelihood than our simple predictor model. However, contains 14 variables is too large. Examining the variables with the largest coefficients, we stepwise built more complex models, building off the simple predictor model.

[184]:

```
X2 = task2.drop(axis=1, columns=['icu_expire_flag','icustay_id'])
# add intercept constant - column of 1's
X2 = sm.add_constant(X2)

# Using SMOTE to balance data and create 20% mortality
smote = SMOTE(sampling_strategy=0.2,random_state=0)
X_resample, y_resample = smote.fit_resample(X2,y)
```

[185]:

```
logistic2_task2 = sm.Logit(endog=y_resample, exog=X_resample).fit(maxiter=10000)
```

```
Warning: Maximum number of iterations has been exceeded.
         Current function value: 0.216252
         Iterations: 10000
/Users/joshbryden/opt/miniconda3/envs/capstone/lib/python3.7/site-
packages/statsmodels/base/model.py:568: ConvergenceWarning: Maximum Likelihood
optimization failed to converge. Check mle_retvals
  ConvergenceWarning)
```

[186]:

```
print(logistic2_task2.summary())
```

```
                           Logit Regression Results
==============================================================================
Dep. Variable:        icu_expire_flag   No. Observations:                30793
Model:                          Logit   Df Residuals:                    30778
```

```
Method:                           MLE   Df Model:                           14
Date:                Sun, 09 May 2021   Pseudo R-squ.:                  0.5200
Time:                        18:12:06   Log-Likelihood:                -6659.1
converged:                      False   LL-Null:                       -13874.
Covariance Type:            nonrobust   LLR p-value:                     0.000
================================================================================
========
                          coef    std err          z      P>|z|      [0.025
0.975]
--------------------------------------------------------------------------------
--------
const                  36.1769      1.177     30.732      0.000      33.870
38.484
gender                  0.0376      0.049      0.760      0.447      -0.059
0.135
age_years               0.0257      0.001     20.131      0.000       0.023
0.028
los                    -0.0240      0.003     -7.848      0.000      -0.030
-0.018
gcs                    -2.8704      0.066    -43.794      0.000      -2.999
-2.742
endotrachflag         -10.5500      0.580    -18.188      0.000     -11.687
-9.413
spo2                    0.0047      0.005      0.874      0.382      -0.006
0.015
heartrate               0.0237      0.003      9.012      0.000       0.019
0.029
meanarterialpressure   -0.0684      0.005    -14.414      0.000      -0.078
-0.059
urineoutput            -0.0162      0.001    -19.612      0.000      -0.018
-0.015
positiveculture        -0.5153      0.743     -0.694      0.488      -1.971
0.941
antibiotics_flag      -31.6119   6.34e+04     -0.000      1.000   -1.24e+05
1.24e+05
mechvent_flag           0.7107      0.066     10.804      0.000       0.582
0.840
vasopressor_flag        2.0657      0.055     37.683      0.000       1.958
2.173
weekend_flag           -0.1057      0.060     -1.759      0.079      -0.224
0.012
================================================================================
========

Possibly complete quasi-separation: A fraction 0.19 of observations can be
perfectly predicted. This might indicate that there is complete
quasi-separation. In this case some parameters will not be identified.
```

[187]:
```python
print(f'Odds for weekend admission to ICU :{np.exp(-0.1057)}')
```

```
Odds for weekend admission to ICU :0.8996945159485037
```

Initially the mechanical ventilation flag was added to the model resulting in a lower log likelihood and odds of 91% higher for weekend mortality.

[188]:
```python
# Split dataset into X and Y variables
y = task2['icu_expire_flag']
X = task2.drop(axis=1, columns=['icu_expire_flag','icustay_id'])

# Select data
X3 = X[['weekend_flag', 'mechvent_flag']]
# add intercept constant - column of 1's
X3 = sm.add_constant(X3)

# Using SMOTE to balance data and create 20% mortality
smote = SMOTE(sampling_strategy=0.2,random_state=0)
X_resample, y_resample = smote.fit_resample(X3,y)
```

[189]:
```python
logistic3_task2 = sm.Logit(endog=y_resample, exog=X_resample).fit(maxiter=1000)
```

```
Optimization terminated successfully.
         Current function value: 0.409221
         Iterations 7
```

[190]:
```python
print(logistic3_task2.summary())
```

```
                           Logit Regression Results
==============================================================================
Dep. Variable:        icu_expire_flag   No. Observations:                30793
Model:                          Logit   Df Residuals:                    30790
Method:                           MLE   Df Model:                            2
Date:                Sun, 09 May 2021   Pseudo R-squ.:                 0.09173
Time:                        18:12:06   Log-Likelihood:                -12601.
converged:                       True   LL-Null:                       -13874.
Covariance Type:            nonrobust   LLR p-value:                     0.000
================================================================================
=
                    coef    std err          z      P>|z|      [0.025
0.975]
--------------------------------------------------------------------------------
-
const            -2.6650      0.034    -78.372      0.000      -2.732
-2.598
weekend_flag     -0.0904      0.040     -2.249      0.024      -0.169
-0.012
mechvent_flag     1.6962      0.038     45.176      0.000       1.623
```

```
1.770
================================================================================
=
```

[191]:
```
print(f'Odds for weekend admission to ICU :{np.exp(-0.0904)}')
```

```
Odds for weekend admission to ICU :0.9135656859018669
```

Building upon this model we added the vasopressor flag resulting in odds of 92% for weekend admission for a change in vasopressor and mechanical ventilation flags.

[192]:
```
# Split dataset into X and Y variables
y = task2['icu_expire_flag']
X = task2.drop(axis=1, columns=['icu_expire_flag','icustay_id'])

# Select data
X4 = X[['weekend_flag', 'mechvent_flag', 'vasopressor_flag']]
# add intercept constant - column of 1's
X4 = sm.add_constant(X4)

# Using SMOTE to balance data and create 20% mortality
smote = SMOTE(sampling_strategy=0.2,random_state=0)
X_resample, y_resample = smote.fit_resample(X4,y)
```

[193]:
```
logistic4_task2 = sm.Logit(endog=y_resample, exog=X_resample).fit(maxiter=1000)
```

```
Optimization terminated successfully.
         Current function value: 0.366537
         Iterations 7
```

[194]:
```
print(logistic4_task2.summary())
```

```
                           Logit Regression Results
==============================================================================
Dep. Variable:        icu_expire_flag   No. Observations:                30793
Model:                          Logit   Df Residuals:                    30789
Method:                           MLE   Df Model:                            3
Date:                Sun, 09 May 2021   Pseudo R-squ.:                  0.1865
Time:                        18:12:06   Log-Likelihood:                -11287.
converged:                       True   LL-Null:                       -13874.
Covariance Type:            nonrobust   LLR p-value:                     0.000
================================================================================
====
                       coef    std err          z      P>|z|      [0.025
0.975]
--------------------------------------------------------------------------------
----
const               -2.9516      0.036    -81.658      0.000      -3.022
-2.881
```

```
weekend_flag        -0.0769      0.043     -1.803      0.071      -0.161
0.007
mechvent_flag        1.2374      0.040     30.593      0.000       1.158
1.317
vasopressor_flag     1.7689      0.035     50.015      0.000       1.700
1.838
================================================================================
====
```

[195]:
```
print(f'Odds for weekend admission to ICU :{np.exp(-0.0769)}')
```

```
Odds for weekend admission to ICU :0.9259824472214596
```

Addition of the endotracheal tube flag resulted in an odds of 88% higher for weekend admission for a change in mechanical ventilation, vasopressor and endotrach flag variables. Compared to the previous model the inclusion of the endotracheal tube variable lowered the odds of a weekend admission mortality.

[196]:
```
# Split dataset into X and Y variables
y = task2['icu_expire_flag']
X = task2.drop(axis=1, columns=['icu_expire_flag','icustay_id'])

# Select data
X5 = X[['weekend_flag', 'mechvent_flag', 'vasopressor_flag', 'endotrachflag']]
# add intercept constant - column of 1's
X5 = sm.add_constant(X5)

# Using SMOTE to balance data and create 20% mortality
smote = SMOTE(sampling_strategy=0.2,random_state=0)
X_resample, y_resample = smote.fit_resample(X5,y)
```

[197]:
```
logistic5_task2 = sm.Logit(endog=y_resample, exog=X_resample).fit(maxiter=1000)
```

```
Optimization terminated successfully.
         Current function value: 0.350633
         Iterations 7
```

[198]:
```
print(logistic5_task2.summary())
```

```
                           Logit Regression Results
==============================================================================
Dep. Variable:        icu_expire_flag   No. Observations:                30793
Model:                          Logit   Df Residuals:                    30788
Method:                           MLE   Df Model:                            4
Date:                Sun, 09 May 2021   Pseudo R-squ.:                  0.2218
Time:                        18:12:06   Log-Likelihood:                -10797.
converged:                       True   LL-Null:                       -13874.
Covariance Type:            nonrobust   LLR p-value:                     0.000
================================================================================
```

```
====
                          coef    std err          z      P>|z|      [0.025
0.975]
--------------------------------------------------------------------------------
----
const              -2.9356      0.036    -81.575      0.000      -3.006
-2.865
weekend_flag       -0.1250      0.044     -2.864      0.004      -0.211
-0.039
mechvent_flag       0.4257      0.049      8.632      0.000       0.329
0.522
vasopressor_flag    1.7664      0.037     48.163      0.000       1.695
1.838
endotrachflag       6.1672      0.196     31.439      0.000       5.783
6.552
================================================================================
====
```

[199]:

```
print(f'Odds for weekend admission to ICU :{np.exp(-0.1250)}')
```

```
Odds for weekend admission to ICU :0.8824969025845955
```

To examine this effect further we built upon this model with the addition of the GCS score.

[200]:

```
# Split dataset into X and Y variables
y = task2['icu_expire_flag']
X = task2.drop(axis=1, columns=['icu_expire_flag','icustay_id'])

# Select data
X6 = X[['weekend_flag', 'mechvent_flag', 'vasopressor_flag', 'endotrachflag',␣
 ↪'gcs']]
# add intercept constant - column of 1's
X6 = sm.add_constant(X6)

# Using SMOTE to balance data and create 20% mortality
smote = SMOTE(sampling_strategy=0.2,random_state=0)
X_resample, y_resample = smote.fit_resample(X6,y)
```

[201]:

```
logistic6_task2 = sm.Logit(endog=y_resample, exog=X_resample).fit(maxiter=1000)
```

```
Optimization terminated successfully.
         Current function value: 0.304933
         Iterations 7
```

[202]:

```
print(logistic6_task2.summary())
```

```
                           Logit Regression Results
==============================================================================
Dep. Variable:        icu_expire_flag   No. Observations:                30793
```

```
Model:                          Logit   Df Residuals:                    30787
Method:                           MLE   Df Model:                            5
Date:                Sun, 09 May 2021   Pseudo R-squ.:                  0.3232
Time:                        18:12:06   Log-Likelihood:                -9389.8
converged:                       True   LL-Null:                       -13874.
Covariance Type:            nonrobust   LLR p-value:                     0.000
================================================================================
====
                       coef    std err          z      P>|z|      [0.025
0.975]
--------------------------------------------------------------------------------
----
const               27.7615      0.686     40.448      0.000      26.416
29.107
weekend_flag        -0.1908      0.048     -4.008      0.000      -0.284
-0.097
mechvent_flag        0.6032      0.053     11.370      0.000       0.499
0.707
vasopressor_flag     1.7572      0.040     43.720      0.000       1.678
1.836
endotrachflag      -10.5602      0.431    -24.489      0.000     -11.405
-9.715
gcs                 -2.3456      0.053    -44.661      0.000      -2.449
-2.243
================================================================================
====
```

This resulted an odds of 82% higher in weekend admission mortality, for a change in mechanical ventilation, vasopressor, endotrach and for each point increase on the GCS scale (0-15).

[203]:
```
print(f'Odds for weekend admission to ICU :{np.exp(-0.1908)}')
```

```
Odds for weekend admission to ICU :0.8262978311925774
```

Finally, the addition of the positive culture flag variable was added to the previous model to investigate any effects on weekend admission. This resulted in the final model having an odds of 83% higher for weekend admission mortality for each change in for a change in mechanical ventilation, vasopressor, endotrach, each point increase on the GCS scale and if a positive culture was present. Examining the log likelihood for each sequential model, we see that the addition of subsequent variable has reduced the log likelihood, indicating a better model fit.

[204]:
```
# Split dataset into X and Y variables
y = task2['icu_expire_flag']
X = task2.drop(axis=1, columns=['icu_expire_flag','icustay_id'])

# Select data
X7 = X[['weekend_flag', 'mechvent_flag', 'vasopressor_flag', 'endotrachflag',␣
 ↪'gcs', 'positiveculture']]
# add intercept constant - column of 1's
```

```
X7 = sm.add_constant(X7)

# Using SMOTE to balance data and create 20% mortality
smote = SMOTE(sampling_strategy=0.2,random_state=0)
X_resample, y_resample = smote.fit_resample(X7,y)
```

[205]:
```
logistic7_task2 = sm.Logit(endog=y_resample, exog=X_resample).fit(maxiter=1000)
```

```
Optimization terminated successfully.
         Current function value: 0.304786
         Iterations 7
```

[206]:
```
print(logistic7_task2.summary())
```

```
                           Logit Regression Results
==============================================================================
Dep. Variable:        icu_expire_flag   No. Observations:                30793
Model:                          Logit   Df Residuals:                    30786
Method:                           MLE   Df Model:                            6
Date:                Sun, 09 May 2021   Pseudo R-squ.:                  0.3235
Time:                        18:12:07   Log-Likelihood:                -9385.3
converged:                       True   LL-Null:                       -13874.
Covariance Type:            nonrobust   LLR p-value:                     0.000
================================================================================
====
                      coef    std err          z      P>|z|      [0.025
0.975]
--------------------------------------------------------------------------------
----
const              27.8382      0.687     40.502      0.000      26.491
29.185
weekend_flag       -0.1922      0.048     -4.034      0.000      -0.286
-0.099
mechvent_flag       0.5963      0.053     11.231      0.000       0.492
0.700
vasopressor_flag    1.7654      0.040     43.839      0.000       1.686
1.844
endotrachflag     -10.4876      0.432    -24.251      0.000     -11.335
-9.640
gcs                -2.3508      0.053    -44.701      0.000      -2.454
-2.248
positiveculture    -1.6745      0.650     -2.576      0.010      -2.949
-0.401
================================================================================
====
```

Interestingly, the odds for each variable in the final model shows that the largest odds increase stems from the vasopressor flag with a 584% increase in odds of mortality for each step change in the other variables.

[207]:
```
odds = np.exp(logistic7_task2.params).tolist()
odds
```

[207]:
```
[1230144385247.1917,
 0.8251468636461092,
 1.8154023334619656,
 5.843845347436964,
 2.788118969423727e-05,
 0.09529647950689693,
 0.1873952377289923]
```

[208]:
```
model_params = pd.DataFrame({'variable':
↪['constant','weekend_flag','mechvent_flag','vasopressor_flag','endotrachflag','gcs','positi
↪'odds': odds})
model_params
```

[208]:

	variable	odds
0	constant	1230144385247.19
1	weekend_flag	0.83
2	mechvent_flag	1.82
3	vasopressor_flag	5.84
4	endotrachflag	0.00
5	gcs	0.10
6	positiveculture	0.19

[209]:
```
pred = logistic7_task2.predict(X7)
pred.head()
```

[209]:
```
0    0.02
1    0.05
2    0.06
3    0.03
4    0.09
dtype: float64
```

To examine the model's performance overall in prediction of mortality with the inclusion of the weekend flag variable, we found an 80% accuracy. Further research will be required to determine how this model would work as a predictor for the first research question. However, it is evident that the weekend effect for ICU mortality does exist within our dataset, especially with the inclusion of the variables on mechanical ventilation, vasopressor administering, endotracheal incubation, presence of bacteria in the blood and for each point change in the GCS scoring system.

[210]:
```
for i in range(0, len(pred)):
    predicted_output = pred.replace()
    if pred[i] >= 0.5:
        pred = pred.replace(pred[i], 1)
```

```
    else:
        pred = pred.replace(pred[i], 0)
```

[211]:
```
accuracy = 0
for i in range(0, len(pred)):
    if y_resample[i] == pred[i]:
        accuracy += 1
accuracy/len(y_resample)
```

[211]: 0.8027149027376351

In conclusion, it is possible to define a prediction model to accurately predict mortality within the ICU on the MIMIC-II dataset using a random forest classifier with tuned hyperparameters. Further research will be required to examine the effect of further tuning on validation and similar datasets before use in a clinical setting. However, the model shows promising results that show which treatments, procedures and test results that provide the greatest insight into mortality in hospital. Furthermore, statistical results from the logistic regression model shows that the MIMIC-II dataset does suffer from the 'weekend effect', in that odds of mortality were shown to be significantly higher for weekend admissions as shown through the simple predictor model. Inclusion of key variables surrounding the use of ventilation and breathing equipment alongside the administering of vasopressors, presence of bacteria and levels of consciousness were shown to improve the models overall fit and present themselves as key clinical measurements to assess the weekend effect. Investigation in a practical sense as to why this may be occurring would be of great use to the Beth Israel Deaconess Medical Centre and shows the roles statistical models can play in examining clinical effects.

References

Angus, DC. Linde-Zwirble, WT. Sirio, CA, Rotondi, AJ. Chelluri, L. Newbold, RC. Lave, JR. Pinksy, MR. 1996. The effect of managed care on ICU length of stay: implications for medicare. Journal of the American Medical Association, 276(13) p1075-1082.

Faust, L. Feldman, K. Chawla, NV. 2019. Critical Care, 23, https://doi.org/10.1186/s13054-019-2479-5.

Giraud ,T. Dhainaut, JF. Vaxelaire, JF. Joseph, T. Journois, D. Bleichner, G. Sollet, JP. Chevret, S. Monsallier, JF. 1993. Iatrogenic complications in adult intensive care units: a prospective two-center study. Critical Care Medicine, 21(1) p40-51.

Hicks, P. Huckson, S. Fenney, E. Leggett, I. Pilcher, D. Littion, E. 2019. The financial cost of intensive care in Australia: a multicentre registry study. Medicine Journal of Australia, 211(7) p324-325. Hui, D. dos Santos, R. Chisholm, G. Bansal, S. Silva, TB. Kilgore, K. Crovador, CS, Yu, X. Swartz, MD. Perez-Cruz, PE. Leite, R. Nascimento, MS. Reddy, S. Seriaco, F. Yennu, S. Paiva, CE. Dev, R. Hall, S. Fajardo, J. Bruera, E. 2014. Clinical signs of impending death in cancer patients. The Oncologist, 19(6), p681–687.

Khan, I & RIDley, S. 2014. Intensive care: who benefits? The Intensive Care Society, 15(4).

Wu, AW. Pronovost, P. Morlock, L. 2002. ICU incIDent reporting systems. Journal of Critical Care, 17(2) p86-94.