Patient Survival Prediction using Deep Learning

Contents

Importing libraries

In [1]:
```
# File system manangement
import time, psutil, os

# Mathematical functions
import math

# Data manipulation
import numpy as np
import pandas as pd

# Plotting and visualization
import matplotlib.pyplot as plt
import matplotlib.patches as mpatches
import seaborn as sns
sns.set_theme()
import plotly.express as px
from plotly.subplots import make_subplots
import plotly.graph_objects as go

# Train-test split and k-fold cross validation
from sklearn.model_selection import train_test_split, KFold,  GridSearchCV

# Missing data imputation
from sklearn.impute import SimpleImputer

# Categorical data encoding
from sklearn.preprocessing import LabelEncoder

# Deep learning
import tensorflow as tf
from tensorflow import keras
from keras import layers
from keras.models import Sequential
from keras.layers import Dense
from keras.optimizers import Adam

# Hyperparameter Tuning
!pip install -q -U keras-tuner
import keras_tuner as kt
from keras_tuner import HyperModel, Hyperband

# Model evaluation
from sklearn import metrics
from sklearn.metrics import accuracy_score
from sklearn.metrics import roc_auc_score
from sklearn.metrics import precision_score
from sklearn.metrics import recall_score
from sklearn.metrics import f1_score

# Model loading
from keras.models import load_model

# Explainable AI
!pip install shap
import shap

# Warning suppression
import warnings
```

```
warnings.filterwarnings('ignore')
```

```
     ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 129.1/129.1 kB 1.0 MB/s eta
0:00:00ta 0:00:01
Collecting shap
  Downloading shap-0.45.1-cp310-cp310-manylinux_2_12_x86_64.manylinux2010_
x86_64.manylinux_2_17_x86_64.manylinux2014_x86_64.whl (540 kB)
     ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 540.5/540.5 kB 3.6 MB/s eta
0:00:00
Requirement already satisfied: numpy in /usr/local/lib/python3.10/dist-pac
kages (from shap) (1.25.2)
Requirement already satisfied: scipy in /usr/local/lib/python3.10/dist-pac
kages (from shap) (1.11.4)
Requirement already satisfied: scikit-learn in /usr/local/lib/python3.10/d
ist-packages (from shap) (1.2.2)
Requirement already satisfied: pandas in /usr/local/lib/python3.10/dist-pa
ckages (from shap) (2.0.3)
Requirement already satisfied: tqdm>=4.27.0 in /usr/local/lib/python3.10/d
ist-packages (from shap) (4.66.4)
Requirement already satisfied: packaging>20.9 in /usr/local/lib/python3.1
0/dist-packages (from shap) (24.0)
Collecting slicer==0.0.8 (from shap)
  Downloading slicer-0.0.8-py3-none-any.whl (15 kB)
Requirement already satisfied: numba in /usr/local/lib/python3.10/dist-pac
kages (from shap) (0.58.1)
Requirement already satisfied: cloudpickle in /usr/local/lib/python3.10/di
st-packages (from shap) (2.2.1)
Requirement already satisfied: llvmlite<0.42,>=0.41.0dev0 in /usr/local/li
b/python3.10/dist-packages (from numba->shap) (0.41.1)
Requirement already satisfied: python-dateutil>=2.8.2 in /usr/local/lib/py
thon3.10/dist-packages (from pandas->shap) (2.8.2)
Requirement already satisfied: pytz>=2020.1 in /usr/local/lib/python3.10/d
ist-packages (from pandas->shap) (2023.4)
Requirement already satisfied: tzdata>=2022.1 in /usr/local/lib/python3.1
0/dist-packages (from pandas->shap) (2024.1)
Requirement already satisfied: joblib>=1.1.1 in /usr/local/lib/python3.10/
dist-packages (from scikit-learn->shap) (1.4.2)
Requirement already satisfied: threadpoolctl>=2.0.0 in /usr/local/lib/pyth
on3.10/dist-packages (from scikit-learn->shap) (3.5.0)
Requirement already satisfied: six>=1.5 in /usr/local/lib/python3.10/dist-
packages (from python-dateutil>=2.8.2->pandas->shap) (1.16.0)
Installing collected packages: slicer, shap
Successfully installed shap-0.45.1 slicer-0.0.8
```

In [2]:
```
# Recording the starting time, which will be complemented with a time check
start = time.time()
```

1. Introduction

Data

In [3]:
```
# Loading the data
data = pd.read_csv('Dataset.csv')

# Memory usage
print("Memory usage: {0:.2f} MB".format(data.memory_usage().sum()/(1024*102

# Printing the dataframe
data
```

```
Memory usage: 130.15 MB
```

Out[3]:

	encounter_id	patient_id	hospital_id	hospital_death	age	bmi	elective_surgery
0	66154	25312	118	0	68.0	22.730000	0
1	114252	59342	81	0	77.0	27.420000	0
2	119783	50777	118	0	25.0	31.950000	0
3	79267	46918	118	0	81.0	22.640000	1
4	92056	34377	33	0	19.0	NaN	0
...	...	...	...	...	...	...	...
91708	91592	78108	30	0	75.0	23.060250	0
91709	66119	13486	121	0	56.0	47.179671	0
91710	8981	58179	195	0	48.0	27.236914	0
91711	33776	120598	66	0	NaN	23.297481	0
91712	1671	53612	104	0	82.0	22.031250	1

91713 rows × 186 columns

Project Objective

The medical records of a patient play a significant role in determining his/her survival odds to a substantial extent. In this project, the objective is to predict whether a patient will survive or not, based on various relevant medical information.

2. Exploratory Data Analysis

- Understanding Features
- Basic Data Exploration
- Univariate Analysis
- Multivariate Analysis

2.1. Understanding Features

2.2. Basic Data Exploration

In [4]:
```
# Shape of the data
print(pd.Series({"Shape of the dataset": data.shape}).to_string())
```

```
Shape of the dataset    (91713, 186)
```

In [5]:
```
# Count of observations
print(pd.Series({"Number of observations in the dataset": len(data)}).to_st
```

```
Number of observations in the dataset    91713
```

In [6]:
```
# Count of columns
print(pd.Series({"Total number of columns in the dataset": len(data.columns
```

```
Total number of columns in the dataset    186
```

In [7]:
```
# Column names
data.columns
```

Out[7]:
```
Index(['encounter_id', 'patient_id', 'hospital_id', 'hospital_death', 'ag
e',
       'bmi', 'elective_surgery', 'ethnicity', 'gender', 'height',
       ...
       'aids', 'cirrhosis', 'diabetes_mellitus', 'hepatic_failure',
       'immunosuppression', 'leukemia', 'lymphoma',
       'solid_tumor_with_metastasis', 'apache_3j_bodysystem',
       'apache_2_bodysystem'],
      dtype='object', length=186)
```

In [8]:
```
# Column datatypes
print(data.dtypes)
```

```
encounter_id                  int64
patient_id                    int64
hospital_id                   int64
hospital_death                int64
age                         float64
                             ...
leukemia                    float64
lymphoma                    float64
solid_tumor_with_metastasis float64
apache_3j_bodysystem         object
apache_2_bodysystem          object
Length: 186, dtype: object
```

In [9]:
```
# Count of column datatypes
cols_int = data.columns[data.dtypes == 'int64'].tolist()
cols_float = data.columns[data.dtypes == 'float64'].tolist()
cols_object = data.columns[data.dtypes == 'object'].tolist()
print(pd.Series({"Number of integer columns": len(cols_int),
                 "Number of float columns": len(cols_float),
                 "Number of object columns": len(cols_object)}).to_string()
```

```
Number of integer columns      8
Number of float columns      170
Number of object columns       8
```

In [10]:
```
# Count of duplicate rows
print(pd.Series({"Number of duplicate rows in the dataset": data.duplicated
```

```
Number of duplicate rows in the dataset    0
```

In [11]:
```
# Count of duplicate columns
print(pd.Series({"Number of duplicate columns in the dataset": data.T.dupli
```

```
Number of duplicate columns in the dataset    3
```

In [12]:
```
# Constant columns
cols_constant = data.columns[data.nunique() == 1].tolist()
if len(cols_constant) == 0:
    cols_constant = "None"
print(pd.Series({"Constant columns in the dataset": cols_constant}).to_stri
```

```
Constant columns in the dataset    [readmission_status]
```

In [13]:
```
# Count of columns with missing values
print(pd.Series({"Number of columns with missing values in the dataset": le
```

```
Number of columns with missing values in the dataset    175
```

In [14]:
```
# Columns with missing values and respective proportions of values that are
print((data.isna().sum()/len(data))[data.isna().sum() != 0].sort_values(asc
```

```
h1_bilirubin_max    0.922650
h1_bilirubin_min    0.922650
h1_lactate_min      0.919924
h1_lactate_max      0.919924
h1_albumin_max      0.913982
                      ...
d1_sysbp_max        0.001734
d1_heartrate_max    0.001581
d1_heartrate_min    0.001581
icu_admit_source    0.001221
gender              0.000273
Length: 175, dtype: float64
```

In [15]:
```
# Statistical description of numerical variables in the dataset
data.describe()
```

Out[15]:

	encounter_id	patient_id	hospital_id	hospital_death	age	b
count	91713.000000	91713.000000	91713.000000	91713.000000	87485.000000	88284.0000
mean	65606.079280	65537.131464	105.669262	0.086302	62.309516	29.1858
std	37795.088538	37811.252183	62.854406	0.280811	16.775119	8.2751
min	1.000000	1.000000	2.000000	0.000000	16.000000	14.8449
25%	32852.000000	32830.000000	47.000000	0.000000	52.000000	23.6419
50%	65665.000000	65413.000000	109.000000	0.000000	65.000000	27.6546
75%	98342.000000	98298.000000	161.000000	0.000000	75.000000	32.9302
max	131051.000000	131051.000000	204.000000	1.000000	89.000000	67.8149

8 rows × 178 columns

In [16]:
```
# Statistical description of categorical variables in the dataset
data.describe(include = ['O'])
```

Out[16]:

	ethnicity	gender	hospital_admit_source	icu_admit_source	icu_stay_type	icu_type
count	90318	91688	70304	91601	91713	91713
unique	6	2	15	5	3	8
top	Caucasian	M	Emergency Department	Accident & Emergency	admit	Med-Surg ICU
freq	70684	49469	36962	54060	86183	50586

In [17]:
```
# Dropping constant columns
data.drop(cols_constant, axis = 1, inplace = True)
cols_int.remove('readmission_status')
```

Dataset synopsis:

- Number of observations: 91713
- Number of columns: 186
- Number of integer columns: 8
- Number of float columns: 170
- Number of object columns: 8
- Number of duplicate observations: 0
- Constant column: `readmission_status`
- Number of columns with missing values: 175
- Columns with over 90% missing values: `h1_bilirubin_max` , `h1_bilirubin_min` , `h1_lactate_min` , `h1_lactate_max` , `h1_albumin_max`
- Memory Usage: 130.15 MB

2.3. Univariate Analysis

- Target variable
- Predictor variables

In general, throughout the notebook, we choose the number of bins of a histogram by the Freedman-Diaconis rule (https://en.wikipedia.org/wiki/Freedman%E2%80%93Diaconis_rule), which suggests the optimal number of bins to grow as $k \sim n^{1/3}$, where n is the total number of observations.

In [18]:
```
# Setting the number of bins
bins_fd = math.floor(len(data)**(1/3))
```

2.3.1. Target variable

The target variable `hospital_death` is a binary variable indicating the survival status of a patient.

$$0 \mapsto \text{patient survived}$$
$$1 \mapsto \text{patient died}$$

In [19]:
```
# Function to construct barplot and donutplot of a dataframe column
def bar_donut(df, col, height = 500, width = 800, manual_title_text = False
    fig = make_subplots(rows = 1, cols = 2, specs = [[{'type': 'xy'}, {'typ
    fig.add_trace(go.Bar(x = df[col].value_counts(sort = False).index.tolis
                         y = df[col].value_counts(sort = False).tolist(),
                         text = df[col].value_counts(sort = False).tolist()
                         textposition = 'auto'),
                         row = 1, col = 1)
    fig.add_trace(go.Pie(values = df[col].value_counts(sort = False).tolist
                         labels = df[col].value_counts(sort = False).index.
                         hole = 0.5, textinfo = 'label+percent', title = f"
                         row = 1, col = 2)
    fig.update_layout(height = height, width = width, showlegend = False,
                      title = {'text': f"Frequency distribution of {col}",
                               'y': 0.95, 'x': 0.5, 'xanchor': 'center', 'y
                      xaxis = dict(tickmode = 'linear', tick0 = 0, dtick =
    if manual_title_text == True:
        fig.update_layout(height = height, width = width, showlegend = Fals
                          title = {'text': title_text,
                                   'y': 0.95, 'x': 0.5, 'xanchor': 'center'
                          xaxis = dict(tickmode = 'linear', tick0 = 0, dtic
    fig.show()
```

In [20]:
```
# hospital_death
bar_donut(data, 'hospital_death')
```

Observation: The dataset is imbalanced with respect to the target variable

2.3.2 Predictor variables

In [21]:
```
# Function to compute almost constant columns and relative frequencies of c
def almost_constant(df, threshold=0.9, show=True, return_list=True):
    rel_freq = []
    for col in df.columns:
        if df[col].notnull().any() and len(df[col].unique()) > 0:  # Check
            mode_rel_freq = max(df[col].value_counts(sort=True) / len(df[co
            if mode_rel_freq > threshold:
                rel_freq.append((col, mode_rel_freq))
    keys = [item[0] for item in rel_freq]
    values = [item[1] for item in rel_freq]
    series = pd.Series(data=values, index=keys).sort_values(ascending=False
    if show:
        print(series.to_string())
    if return_list:
        return keys
```

In [22]:
```
# Almost constant columns and relative frequencies of corresponding modes
almost_constant_cols = almost_constant(data.drop('hospital_death', axis = 1
```

```
aids                            0.991353
lymphoma                        0.988104
leukemia                        0.985193
hepatic_failure                 0.979316
gcs_unable_apache               0.979272
cirrhosis                       0.976634
solid_tumor_with_metastasis     0.971727
immunosuppression               0.966243
arf_apache                      0.964443
icu_stay_type                   0.939703
```

Observations:

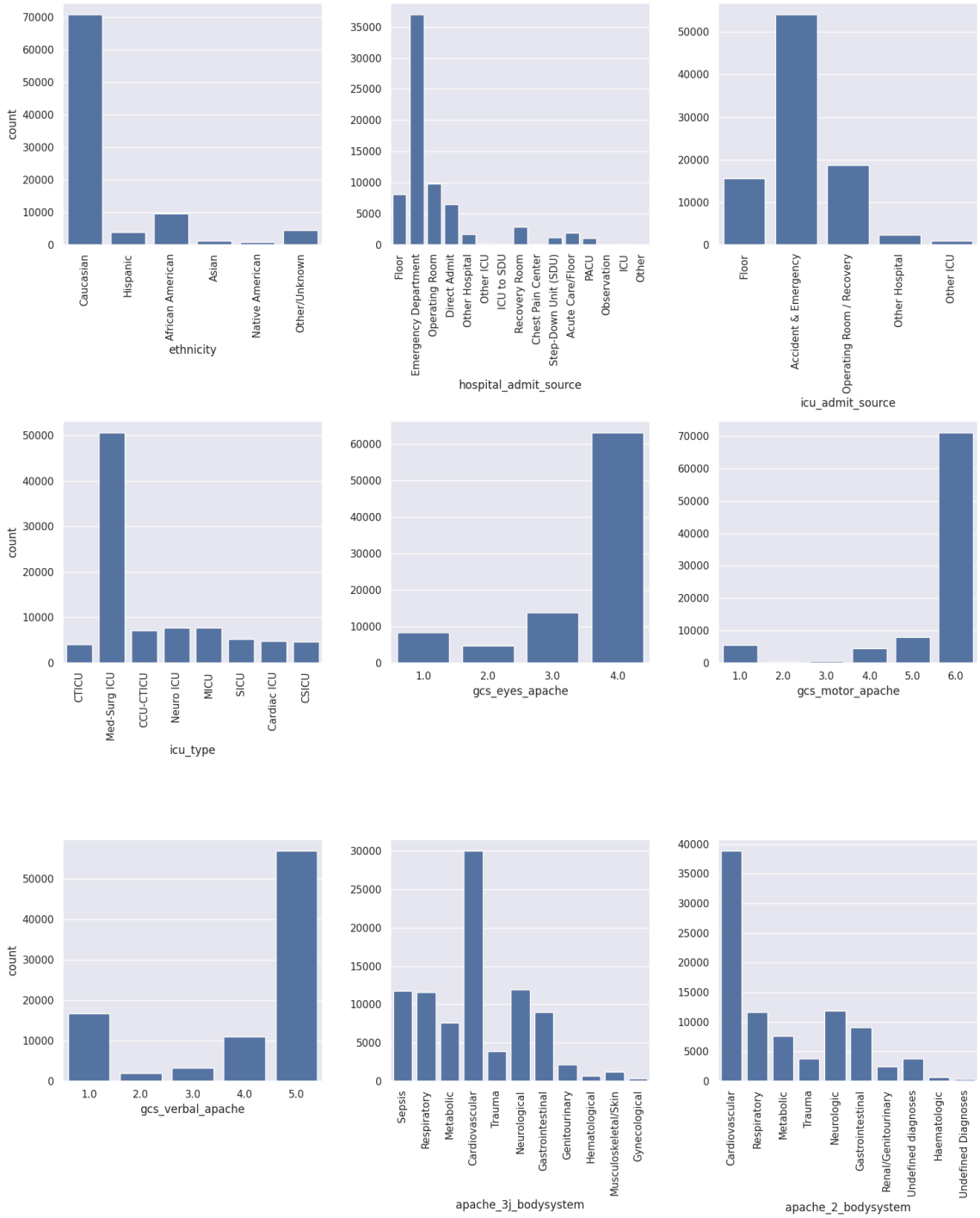
- The column `readmission_status` is same for every observation, and hence is not relevant in the context of predicting the target variable
- Apart from `readmission_status`, relative frequency of the mode value is over 0.9 for 10 other predictor variables, among which `icu_stay_type` is an object variable, taking values `admit`, `transfer` and `readmit`, while the rest are binary float variables, taking values `0.0` and `1.0`

In [23]:
```
# Donutplots in N x 3 grid form
def donuts_grid(df, cols, ncols = 3, hole = 0.5, height = 1500, width = 900
    nrows = math.ceil(len(cols)/ncols)
    specs = np.full((nrows, ncols), {'type': 'domain'}).tolist()
    fig = make_subplots(rows = nrows, cols = ncols, specs = specs)
    count = 0
    break_flag = False
    for row in range(nrows):
        for col in range(ncols):
            i = (row * ncols) + col
            fig.add_trace(go.Pie(values = df[cols[i]].value_counts(sort = F
                                 labels = df[cols[i]].value_counts(sort = F
                                 hole = hole, textinfo = 'percent', title =
                          row = row + 1, col = col + 1)
            count = count + 1
            if count == len(cols):
                break_flag = True
                break

        if break_flag == True:
            break

    fig.update_layout(height = height, width = width)
    fig.show()
```

In [24]:
```
# Binary columns except gender and hospital_death
cols_binary = [col for col in data.columns if data[col].nunique() == 2]
cols_binary.remove('gender')
cols_binary.remove('hospital_death')
donuts_grid(data, cols_binary, ncols = 4, hole = 0.5, height = 1250, width
```

In [25]:
```
# gender and icu_stay_type
fig = make_subplots(rows = 1, cols = 2, specs = [[{'type': 'domain'}, {'typ
fig.add_trace(go.Pie(values = data['gender'].value_counts(sort = False).tol
                     labels = data['gender'].value_counts(sort = False).ind
                     hole = 0.5, textinfo = 'label+percent', title = 'gende
                     row = 1, col = 1)
fig.add_trace(go.Pie(values = data['icu_stay_type'].value_counts(sort = Fal
                     labels = data['icu_stay_type'].value_counts(sort = Fal
                     hole = 0.5, textinfo = 'label+percent', title = 'icu_s
                     row = 1, col = 2)
fig.update_layout(height = 450, width = 900, showlegend = False)
fig.show()
```

In [26]:
```
# Columns with 4 to 15 distinct values
cols_selected = [col for col in data.columns if 3 < data[col].nunique() <=
col_vert = ['ethnicity', 'hospital_admit_source', 'icu_admit_source', 'icu_
nrows = math.ceil(len(cols_selected)/3)
fig, ax = plt.subplots(nrows, 3, figsize = (15, 6.2 * nrows), sharey = Fals
for i in range(len(cols_selected)):
    countplot = sns.countplot(data = data, x = cols_selected[i], ax = ax[i/
    if cols_selected[i] in col_vert:
        countplot.set_xticklabels(countplot.get_xticklabels(), rotation = 9
    if i%3 != 0:
        ax[i//3, i%3].set_ylabel(" ")
plt.tight_layout()
plt.show()
```



In [27]:
```
# Non-float columns with more than 15 distinct values
[col for col in data.columns if data[col].nunique() > 15 and data[col].dtyp
```

Out[27]:
```
['encounter_id', 'patient_id', 'hospital_id', 'icu_id']
```

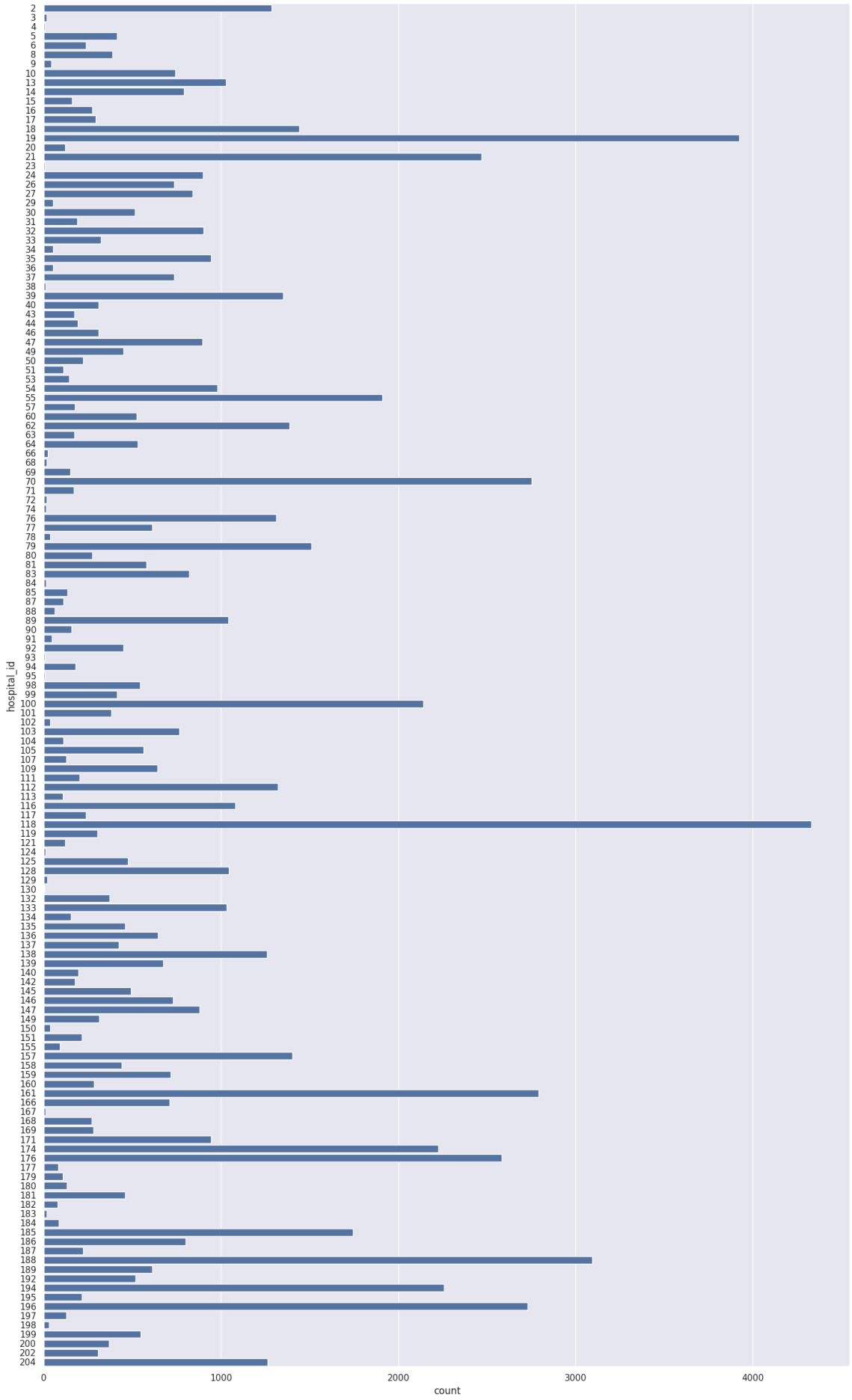
In [28]:
```
# Number of unique values
keys = ['encounter_id', 'patient_id', 'hospital_id', 'icu_id']
values = [data[col].nunique() for col in keys]
print(pd.Series(data = values, index = keys).to_string())
```

```
encounter_id    91713
patient_id      91713
hospital_id       147
icu_id            241
```

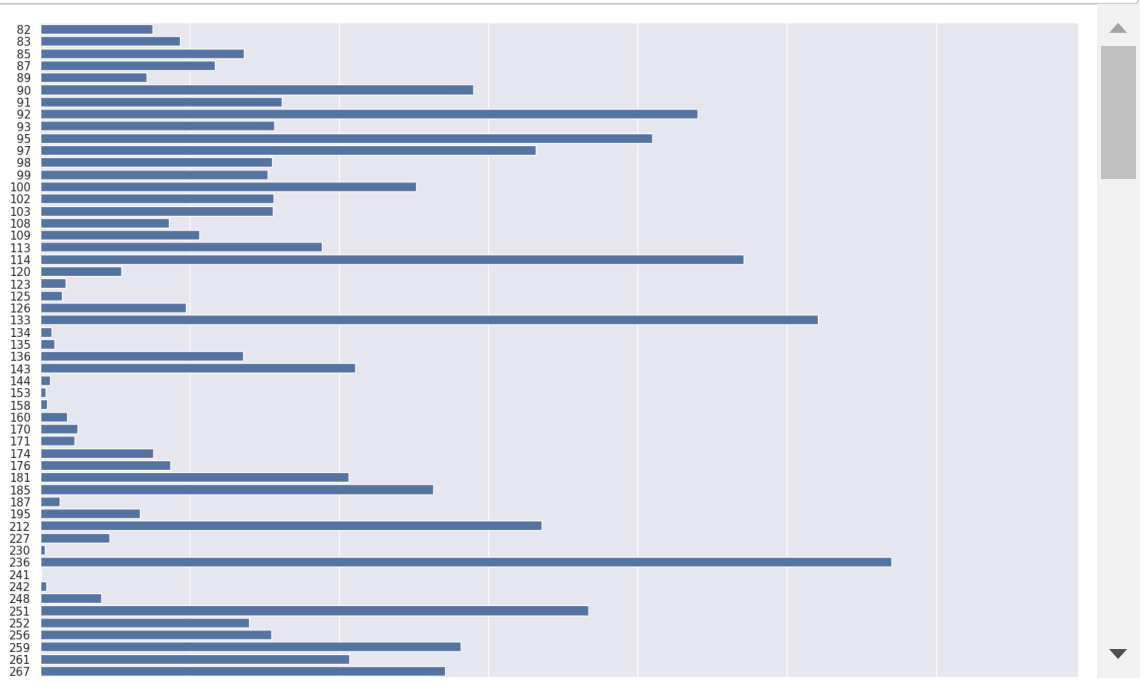
Observation: The features `encounter_id` and `patient_id` are unique for each observation, and hence do not contribute to the task of predicting the target variable.

In [29]:
```
# Dropping encounter_id and patient_id
data.drop(['encounter_id', 'patient_id'], axis = 1, inplace = True)
cols_int.remove('encounter_id')
cols_int.remove('patient_id')
```

In [30]:
```
# hospital_id
plt.figure(figsize = (15, data['hospital_id'].nunique()/6))
sns.countplot(data = data, y = 'hospital_id')
plt.tight_layout()
plt.show()
```

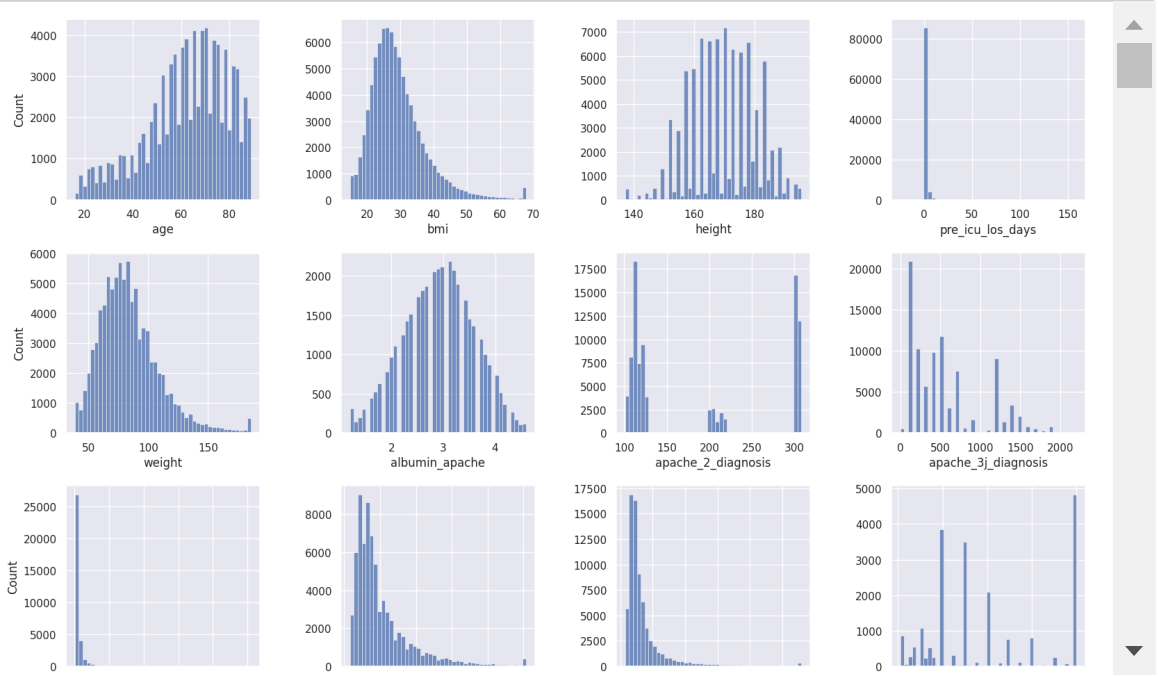


In [31]:
```
# icu_id
plt.figure(figsize = (15, data['icu_id'].nunique()/6))
sns.countplot(data = data, y = 'icu_id')
plt.tight_layout()
plt.show()
```



In [32]:
```
# Histograms in grid
def distribution_plot(df, cols, ncols = 4, kind = 'hist', hue = None, heigh
    cols = [col for col in df.columns if df[col].nunique() > 15 and df[col]
    bins_fd = math.floor(len(df)**(1/3))
    nrows = math.ceil(len(cols)/ncols)
    if kind == 'hist':
        fig, ax = plt.subplots(nrows, ncols, figsize = (width*ncols, height
        for i in range(len(cols)):
            sns.histplot(data = df, x = cols[i], bins = bins_fd, hue = hue,
            if i % ncols != 0:
                ax[i // ncols, i % ncols].set_ylabel(" ")
    elif kind == 'kde':
        fig, ax = plt.subplots(nrows, ncols, figsize = (width*ncols, height
        for i in range(len(cols)):
            sns.kdeplot(data = df, x = cols[i], hue = hue, ax = ax[i // nco
            if i % ncols != 0:
                ax[i // ncols, i % ncols].set_ylabel(" ")
    else:
        raise TypeError(f"'{kind}' is not a valid argument for the paramete
    plt.tight_layout()
    plt.show()
```

In [33]:
```
# Float columns with more than 15 distinct values
cols_selected = [col for col in data.columns if data[col].nunique() > 15 an
distribution_plot(df = data, cols = cols_selected, kind = 'hist')
```



2.4. Multivariate Analysis

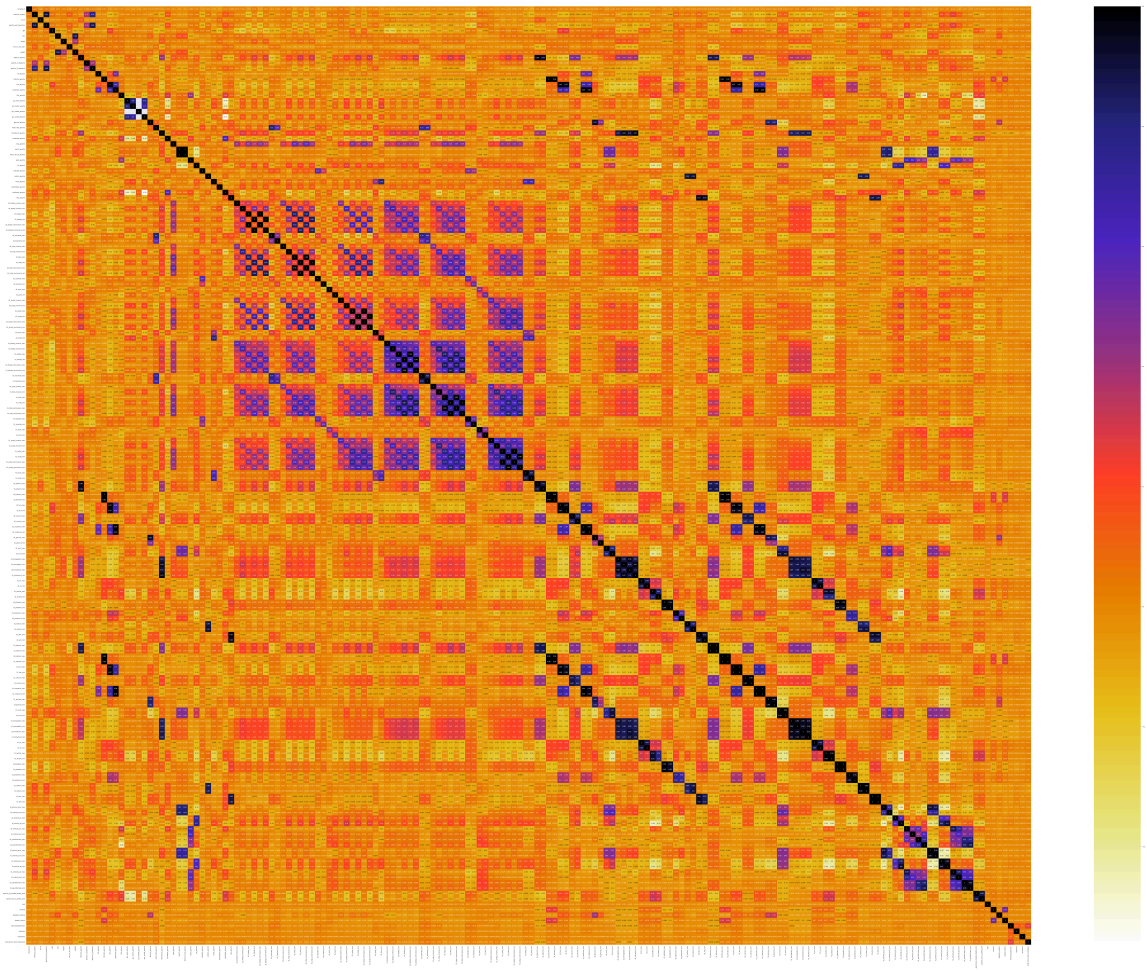
- Relationships among the predictor variables
- Relationships of the target variable with the predictor variables

2.4.1. Relationships among the predictor variables

Correlation structure among numerical features

In [34]:
```
# Heatmap
cols_selected = [col for col in cols_int + cols_float if col != 'hospital_d
plt.figure(figsize = (180, 135))
sns.heatmap(data[cols_selected].corr(), annot = True, cmap = plt.cm.CMRmap_
```

Out[34]: <Axes: >



In [35]:
```
# Function to detect pairs with extreme correlation
def pairs_with_strong_corr(df, cols, threshold = 0.8, show = True, return_v
    variable_list = []
    corr_positive_list = []
    corr_negative_list = []
    for i in range(len(cols)):
        for j in range(len(cols)):
            if i<j:
                corr = df[cols[i]].corr(df[cols[j]])
                if corr > threshold:
                    variable_list = variable_list + [cols[i], cols[j]]
                    corr_positive_list.append(((cols[i], cols[j]), corr))
                if corr < -threshold:
                    variable_list = variable_list + [cols[i], cols[j]]
                    corr_negative_list.append(((cols[i], cols[j]), corr))
    if show == True:
        corr_positive = pd.Series(data = [item[1] for item in corr_positive
        corr_negative = pd.Series(data = [item[1] for item in corr_negative
        print("Pairs with extreme positive correlation:")
        print(" ")
        print(corr_positive.sort_values(ascending = False).to_string())
        print(" ")
        print("Pairs with extreme negative correlation:")
        print(" ")
        print(corr_negative.sort_values(ascending = True).to_string())
    if return_variables == True:
        return variable_list
```

In [36]:
```
# Detecting pairs with extreme correlation
cols_selected = [col for col in cols_int + cols_float if col != 'hospital_d
pairs_with_strong_corr(df = data, cols = cols_selected, threshold = 0.9)
```

Pairs with extreme positive correlation:

(d1_inr_min, h1_inr_min)	1.000000
(paco2_apache, paco2_for_ph_apache)	1.000000
(d1_inr_max, h1_inr_max)	1.000000
(h1_bilirubin_max, h1_bilirubin_min)	0.999934
(h1_albumin_max, h1_albumin_min)	0.999727
(h1_bun_max, h1_bun_min)	0.999630
(h1_creatinine_max, h1_creatinine_min)	0.999405
(h1_platelets_max, h1_platelets_min)	0.998206
(d1_diasbp_max, d1_diasbp_noninvasive_max)	0.997831
(h1_wbc_max, h1_wbc_min)	0.997524
(d1_diasbp_min, d1_diasbp_noninvasive_min)	0.996871
(h1_mbp_min, h1_mbp_noninvasive_min)	0.996673
(bilirubin_apache, d1_bilirubin_max)	0.996568
(d1_sysbp_max, d1_sysbp_noninvasive_max)	0.996560
(h1_sysbp_max, h1_sysbp_noninvasive_max)	0.996377
(d1_sysbp_min, d1_sysbp_noninvasive_min)	0.996322
(d1_mbp_min, d1_mbp_noninvasive_min)	0.995532
(creatinine_apache, d1_creatinine_max)	0.993802
(h1_hco3_max, h1_hco3_min)	0.993134
(d1_creatinine_max, h1_creatinine_max)	0.988856
(d1_creatinine_max, h1_creatinine_min)	0.988305
(d1_bilirubin_max, d1_bilirubin_min)	0.988069
(h1_sysbp_min, h1_sysbp_noninvasive_min)	0.988018
(bilirubin_apache, d1_bilirubin_min)	0.987723
(h1_lactate_max, h1_lactate_min)	0.986786
(h1_mbp_max, h1_mbp_noninvasive_max)	0.986584
(bun_apache, d1_bun_max)	0.986565
(d1_bilirubin_min, h1_bilirubin_min)	0.985227
(d1_bilirubin_min, h1_bilirubin_max)	0.985192
(d1_mbp_max, d1_mbp_noninvasive_max)	0.984703
(h1_diasbp_max, h1_diasbp_noninvasive_max)	0.984292
(d1_bilirubin_max, h1_bilirubin_max)	0.984276
(d1_bilirubin_max, h1_bilirubin_min)	0.984242
(bilirubin_apache, h1_bilirubin_max)	0.982909
(bilirubin_apache, h1_bilirubin_min)	0.982885
(h1_diasbp_min, h1_diasbp_noninvasive_min)	0.981733
(d1_bun_max, h1_bun_max)	0.981409
(d1_bun_max, h1_bun_min)	0.981044
(d1_platelets_max, h1_platelets_max)	0.980230
(creatinine_apache, h1_creatinine_min)	0.978987
(creatinine_apache, h1_creatinine_max)	0.978634
(h1_sodium_max, h1_sodium_min)	0.978568
(d1_platelets_max, h1_platelets_min)	0.978348
(h1_calcium_max, h1_calcium_min)	0.976344
(bun_apache, h1_bun_min)	0.973320
(bun_apache, h1_bun_max)	0.973104
(creatinine_apache, d1_creatinine_min)	0.970275
(d1_creatinine_max, d1_creatinine_min)	0.969980
(wbc_apache, d1_wbc_max)	0.969516
(hematocrit_apache, d1_hematocrit_min)	0.968660
(bun_apache, d1_bun_min)	0.967500
(d1_bun_max, d1_bun_min)	0.967480
(d1_hemaglobin_min, d1_hematocrit_min)	0.965184
(h1_hematocrit_max, h1_hematocrit_min)	0.965069
(h1_glucose_max, h1_glucose_min)	0.963623
(d1_platelets_max, d1_platelets_min)	0.963462
(albumin_apache, d1_albumin_max)	0.961319
(h1_hemaglobin_max, h1_hemaglobin_min)	0.959778
(h1_hemaglobin_max, h1_hematocrit_max)	0.958879

```
(h1_hemaglobin_min, h1_hematocrit_min)          0.958821
(d1_hemaglobin_max, d1_hematocrit_max)          0.955264
(h1_arterial_pco2_max, h1_arterial_pco2_min)    0.953722
(d1_creatinine_min, h1_creatinine_min)          0.952895
(d1_creatinine_min, h1_creatinine_max)          0.952290
(h1_potassium_max, h1_potassium_min)            0.948954
(h1_pao2fio2ratio_max, h1_pao2fio2ratio_min)    0.948134
(d1_bun_min, h1_bun_min)                        0.947840
(d1_bun_min, h1_bun_max)                        0.947419
(hematocrit_apache, d1_hemaglobin_min)          0.945210
(h1_hemaglobin_min, h1_hematocrit_max)          0.944205
(d1_platelets_min, h1_platelets_min)            0.940831
(h1_arterial_ph_max, h1_arterial_ph_min)        0.939925
(d1_platelets_min, h1_platelets_max)            0.938986
(albumin_apache, d1_albumin_min)                0.938142
(h1_hemaglobin_max, h1_hematocrit_min)          0.937758
(d1_hematocrit_max, h1_hematocrit_max)          0.933480
(d1_wbc_max, h1_wbc_max)                        0.930459
(wbc_apache, d1_wbc_min)                        0.929256
(d1_wbc_max, h1_wbc_min)                        0.928045
(d1_lactate_max, h1_lactate_max)                0.925609
(d1_lactate_max, h1_lactate_min)                0.924027
(glucose_apache, d1_glucose_max)                0.923000
(sodium_apache, d1_sodium_min)                  0.919429
(d1_hco3_min, h1_hco3_min)                      0.918726
(d1_hemaglobin_max, h1_hemaglobin_max)          0.918192
(d1_wbc_max, d1_wbc_min)                        0.918107
(d1_arterial_pco2_max, h1_arterial_pco2_max)    0.912731
(d1_hco3_min, h1_hco3_max)                      0.912198
(d1_albumin_max, d1_albumin_min)                0.912165
(d1_hematocrit_max, h1_hematocrit_min)          0.911237
(hematocrit_apache, d1_hematocrit_max)          0.909753
(elective_surgery, apache_post_operative)       0.908247
(d1_sodium_min, h1_sodium_min)                  0.901535
(d1_hematocrit_max, d1_hematocrit_min)          0.901398

Pairs with extreme negative correlation:

Series([], )
```

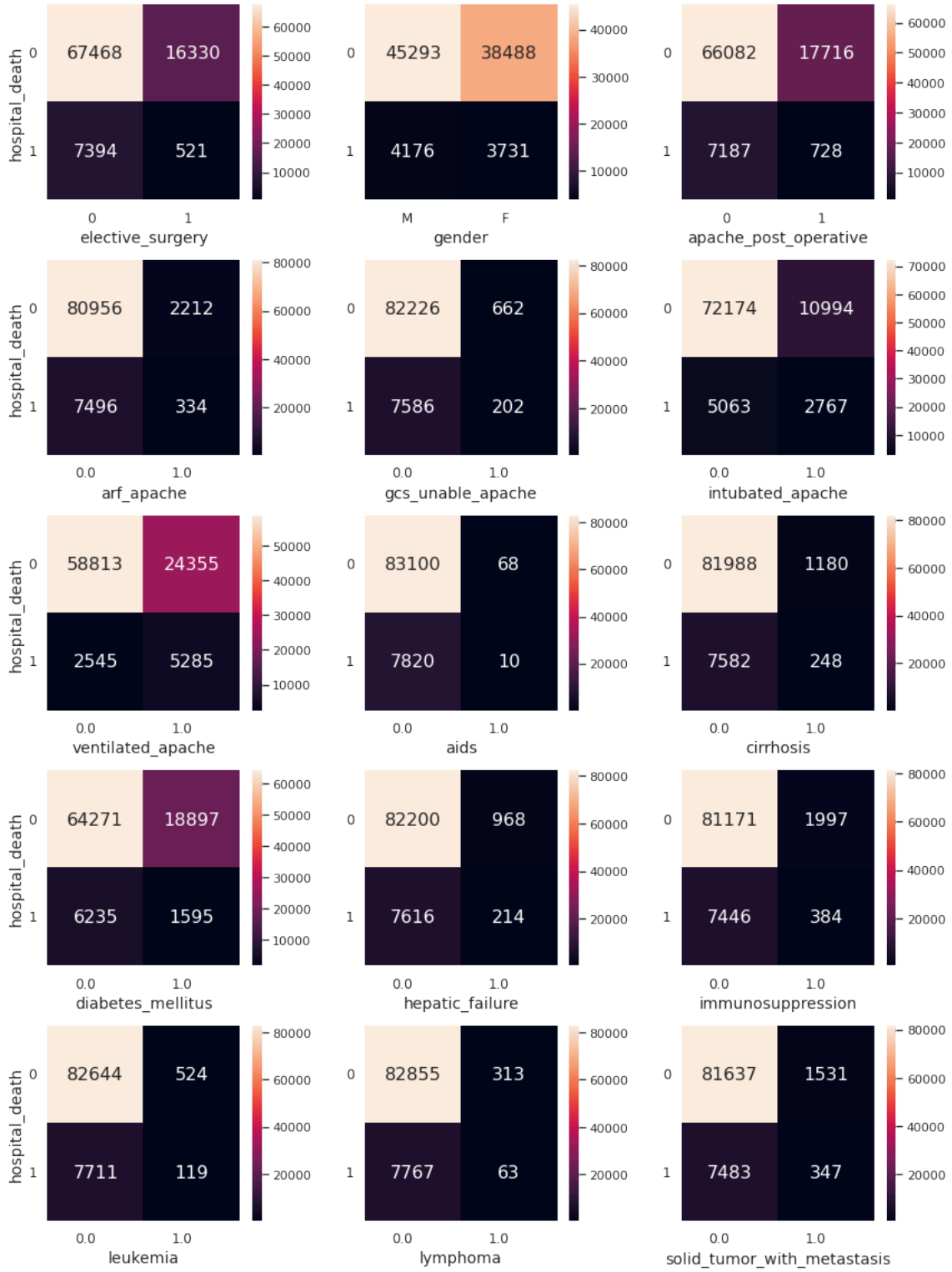
Observations:

- There are 94 pairs of numerical features with correlation coefficient over 0.9
- There are no pair of numerical features with correlation coefficient below -0.9
- 3 pairs of numerical features have correlation coefficient exactly equal to 1, i.e. there exists perfect linear relationship (with positive slope) between the variables in each of those pairs (this corresponds to the fact that there are 3 duplicate columns in the dataset)

2.4.2. Relationships of the target variable with the predictor variables

In [37]:
```
# Contingency tables for target variable and binary features
def contingency_binary(df, target, ncols = 3, figsize_multiplier = 2):
    cols_binary = [col for col in df.columns if df[col].nunique() == 2]
    cols_binary.remove(target)
    if len(cols_binary) == 0:
        print("The dataset does not contain a binary feature")
    else:
        nrows = math.ceil(len(cols_binary)/ncols)
        nvals = 2 # Binary variables
        figsize = (figsize_multiplier*nvals*ncols, 0.8*figsize_multiplier*n
        fig, ax = plt.subplots(nrows, ncols, figsize = figsize, sharey = Fa
        class_names_2 = df[target].value_counts().index.tolist()
        for i in range(len(cols_binary)):
            class_names_1 = df[cols_binary[i]].value_counts().index.tolist(
            contingency_mat = np.zeros(shape = (len(class_names_2), len(cla
            for j in range(len(class_names_2)):
                for k in range(len(class_names_1)):
                    contingency_mat[j][k] = len([l for l in range(len(df))
            contingency_table_df = pd.DataFrame(contingency_mat)
            hm = sns.heatmap(contingency_table_df, annot = True, annot_kws
            hm.set_xlabel(f'{cols_binary[i]}', fontsize = 14)
            hm.set_ylabel(target, fontsize = 14)
            hm.set_xticklabels(class_names_1, fontdict = {'fontsize': 12},
            hm.set_yticklabels(class_names_2, fontdict = {'fontsize': 12},
            if i % ncols != 0:
                ax[i // ncols, i % ncols].set_ylabel(" ")
        plt.tight_layout()
        plt.show()
```

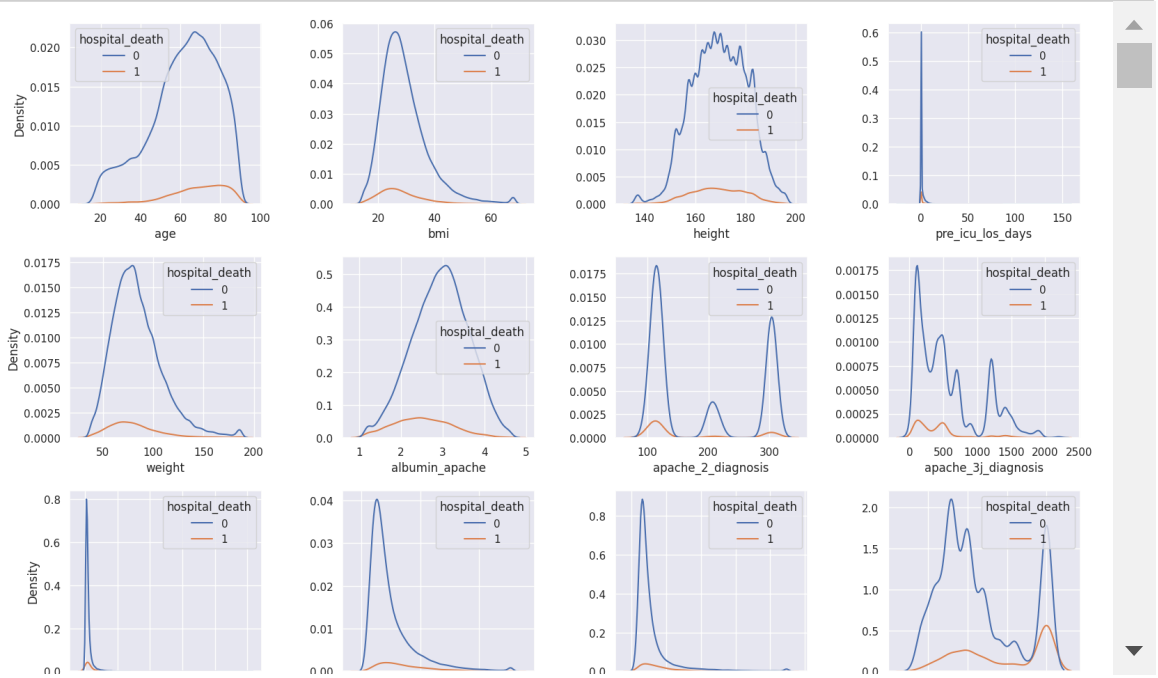
In [ ]:
```
# Target x Binary features
contingency_binary(df = data, target = 'hospital_death', ncols = 3, figsize
```



In [38]:
```
# Contingency table for target variable and general categorical feature
def contingency_table(df, feature, target, figsize_multiplier = 2, title =
    class_names_1 = df[feature].value_counts().index.tolist()
    class_names_2 = df[target].value_counts().index.tolist()

    contingency_mat = np.zeros(shape = (len(class_names_2), len(class_names
    for i in range(len(class_names_2)):
        for j in range(len(class_names_1)):
            contingency_mat[i][j] = len([k for k in range(len(df)) if df[ta

    contingency_table_df = pd.DataFrame(contingency_mat)
    plt.figure(figsize = (figsize_multiplier*len(class_names_1), 0.8*figsiz
    if title == True:
        plt.title(f"{target} x {feature}")
    hm = sns.heatmap(contingency_table_df, annot = True, annot_kws = {"size
    hm.set_xlabel(f'{feature}', fontsize = 14)
    hm.set_ylabel(f'{target}', fontsize = 14)
    hm.set_xticklabels(class_names_1, fontdict = {'fontsize': 12}, rotation
    hm.set_yticklabels(class_names_2, fontdict = {'fontsize': 12}, rotation
    plt.grid(False)
    plt.show()
```

In [39]:
```
# Target x Features taking 3 to 15 distinct values
cols_selected = [col for col in data.columns if 3 <= data[col].nunique() <=
cols_xticklabels = ['ethnicity', 'hospital_admit_source', 'icu_admit_source
for col in cols_selected:
    if col in cols_xticklabels:
        rotate_xticklabels = 45
    else:
        rotate_xticklabels = 0
    contingency_table(df = data, feature = col, target = 'hospital_death',
```

In [40]:
```
# Target x Float features with more than 15 distinct values
cols_selected = [col for col in data.columns if data[col].nunique() > 15 an
distribution_plot(df = data, cols = cols_selected, kind = 'kde', hue = 'hos
```



In [41]:
```
data.head()
```

Out[41]:

	hospital_id	hospital_death	age	bmi	elective_surgery	ethnicity	gender	height	hosp
0	118	0	68.0	22.73	0	Caucasian	M	180.3	
1	81	0	77.0	27.42	0	Caucasian	F	160.0	
2	118	0	25.0	31.95	0	Caucasian	F	172.7	Eme
3	118	0	81.0	22.64	1	Caucasian	F	165.1	
4	33	0	19.0	NaN	0	Caucasian	M	188.0	

5 rows × 183 columns

In [43]:
```
data.info(verbose=True)
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 91713 entries, 0 to 91712
Data columns (total 183 columns):
 #    Column                          Dtype
---   ------                          -----
 0    hospital_id                     int64
 1    hospital_death                  int64
 2    age                             float64
 3    bmi                             float64
 4    elective_surgery                int64
 5    ethnicity                       object
 6    gender                          object
 7    height                          float64
 8    hospital_admit_source           object
 9    icu_admit_source                object
 10   icu_id                          int64
 11   icu_stay_type                   object
 12   icu_type                        object
 13   pre_icu_los_days                float64
```

In [ ]:

In [ ]:

3. Data Preprocessing

- Train-Test Split
- Missing Data Imputation
- Categorical Data Encoding
- Normalization

3.1. Train-Test Split

In [44]:
```
X = data.drop('hospital_death', axis = 1) # Independent variables
y = data['hospital_death'] # Target variable
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size = 0.2,
```

Training data

In [45]:
```
# Training set - Predictor variables
X_train
```

Out[45]:

	hospital_id	age	bmi	elective_surgery	ethnicity	gender	height	hospital_adr
52236	133	67.0	31.086173	1	African American	F	152.4	Opera
30240	88	29.0	31.210012	0	Caucasian	M	179.0	Emergency D
40425	160	74.0	26.201719	1	Asian	M	167.6	
26444	147	NaN	19.950930	0	Caucasian	M	176.0	D
26777	161	38.0	21.490626	0	Hispanic	F	162.0	Emergency D
...	...	...	...	...	...	...	...	
65294	43	48.0	25.951557	0	Caucasian	M	170.0	
16486	5	88.0	24.263117	0	Hispanic	M	185.4	
1888	83	51.0	43.618040	1	Caucasian	F	157.5	Opera
64377	19	48.0	25.226369	0	Caucasian	F	173.0	D
77555	196	79.0	19.469733	0	African American	M	170.2	Acute

73370 rows × 182 columns

In [46]:
```
X_train.info(verbose=True)
```

```
<class 'pandas.core.frame.DataFrame'>
Index: 73370 entries, 52236 to 77555
Data columns (total 182 columns):
 #    Column                          Dtype
---   ------                          -----
 0    hospital_id                     int64
 1    age                             float64
 2    bmi                             float64
 3    elective_surgery                int64
 4    ethnicity                       object
 5    gender                          object
 6    height                          float64
 7    hospital_admit_source           object
 8    icu_admit_source                object
 9    icu_id                          int64
 10   icu_stay_type                   object
 11   icu_type                        object
 12   pre_icu_los_days                float64
 13   weight                          float64
```

In [47]:
```
# Training set - Target variable
bar_donut(pd.DataFrame(y_train), 'hospital_death',  manual_title_text = Tru
          title_text = "Frequency distribution of hospital_death in the tra
```

Test data

In [48]:
```
# Test set - Predictor variables
X_test
```

Out[48]:

	hospital_id	age	bmi	elective_surgery	ethnicity	gender	height	hospital
48467	21	71.0	27.768126	0	Other/Unknown	F	167.6	
65356	19	86.0	28.581315	0	Caucasian	M	170.0	Emerge
16224	116	80.0	18.966902	0	Hispanic	F	165.1	
44085	186	67.0	32.583562	0	Caucasian	M	185.4	(
79745	103	60.0	27.594032	0	NaN	F	172.7	Emerge
...	...	...	...	...	...	...	...	
51289	133	59.0	21.370140	1	Caucasian	F	162.6	(
37193	70	77.0	31.336438	0	Caucasian	M	169.0	
83377	47	37.0	35.015582	0	Caucasian	F	152.0	Emerge
89064	188	75.0	26.525573	0	Caucasian	F	157.5	(
411	118	37.0	27.223993	0	Hispanic	M	180.3	Emerge

18343 rows × 182 columns

In [49]:
```
# Test set - Target variable
bar_donut(pd.DataFrame(y_test), 'hospital_death', manual_title_text = True,
          title_text = "Frequency distribution of hospital_death in the tes
```

3.2. Missing Data Imputation

In [50]:
```
# Count of missing values for the target variable
print(pd.Series({"Number of missing target values in the training set": y_t
                 "Number of missing target values in the test set": y_test.
```

```
Number of missing target values in the training set    0
Number of missing target values in the test set        0
```

Dropping columns with majority of the observations missing

In [51]:
```
# Columns with more than 50% missing values in the training set
print((X_train.isna().sum()/len(X_train))[X_train.isna().sum()/len(X_train)
```

```
h1_bilirubin_max      0.921971
h1_bilirubin_min      0.921971
h1_lactate_min        0.920117
h1_lactate_max        0.920117
h1_albumin_max        0.913343
                          ...
h1_glucose_max        0.574145
h1_glucose_min        0.574145
d1_albumin_min        0.535028
d1_albumin_max        0.535028
urineoutput_apache    0.534823
Length: 74, dtype: float64
```

In [52]:
```
X_train.info(verbose=True)
```

```
<class 'pandas.core.frame.DataFrame'>
Index: 73370 entries, 52236 to 77555
Data columns (total 182 columns):
 #    Column                          Dtype
---   ------                          -----
 0    hospital_id                     int64
 1    age                             float64
 2    bmi                             float64
 3    elective_surgery                int64
 4    ethnicity                       object
 5    gender                          object
 6    height                          float64
 7    hospital_admit_source           object
 8    icu_admit_source                object
 9    icu_id                          int64
 10   icu_stay_type                   object
 11   icu_type                        object
 12   pre_icu_los_days                float64
 13   weight                          float64
```

We drop the 74 features, which have over 50% values missing in the training dataset, from the subsequent analysis.

In [53]:
```
# Dropping columns with more than 50% missing values in the training set
majority_missing = (X_train.isna().sum()/len(X_train))[data.isna().sum()/le
X_train.drop(majority_missing, axis = 1, inplace = True)
X_test.drop(majority_missing, axis = 1, inplace = True)
```

Mode imputation

In [54]:
```
# Function to impute missing values with the most frequent value appearing
def mode_imputer(data):
    data_imputed = data.copy(deep = True)
    imputer = SimpleImputer(strategy = 'most_frequent')
    data_imputed.iloc[:, :] = imputer.fit_transform(data_imputed)
    return data_imputed
```

In [55]:
```
X_train.info(verbose=True)
```

```
<class 'pandas.core.frame.DataFrame'>
Index: 73370 entries, 52236 to 77555
Data columns (total 108 columns):
 #    Column                          Dtype
---   ------                          -----
 0    hospital_id                     int64
 1    age                             float64
 2    bmi                             float64
 3    elective_surgery                int64
 4    ethnicity                       object
 5    gender                          object
 6    height                          float64
 7    hospital_admit_source           object
 8    icu_admit_source                object
 9    icu_id                          int64
 10   icu_stay_type                   object
 11   icu_type                        object
 12   pre_icu_los_days                float64
 13   weight                          float64
```

Proportion-based imputation

With the goal of keeping the feature distributions same before and after imputation, we impute the missing values in a column in such a way so that the proportions of the existing unique values in that particular column remain roughly same as those were prior to the imputation. The following function takes a dataframe, implements the proportion-based imputation in each column containing missing values and returns the resulting dataframe.

In [56]:
```
# Function to impute missing values proportionately with respect to the exi
def prop_imputer(df):
    df_prop = df.copy(deep = True)
    missing_cols = df_prop.isna().sum()[df_prop.isna().sum() != 0].index.to
    for col in missing_cols:
        values_col = df_prop[col].value_counts(normalize = True).index.toli
        probabilities_col = df_prop[col].value_counts(normalize = True).val
        df_prop[col] = df_prop[col].fillna(pd.Series(data = np.random.choic
    return df_prop
```

In [57]:
```
# Proportion-based imputation
X_train = prop_imputer(X_train)
X_test = prop_imputer(X_test)
```

In [58]:
```
X_train.info(verbose=True)
```

```
<class 'pandas.core.frame.DataFrame'>
Index: 73370 entries, 52236 to 77555
Data columns (total 108 columns):
 #    Column                          Dtype
---   ------                          -----
 0    hospital_id                     int64
 1    age                             float64
 2    bmi                             float64
 3    elective_surgery                int64
 4    ethnicity                       object
 5    gender                          object
 6    height                          float64
 7    hospital_admit_source           object
 8    icu_admit_source                object
 9    icu_id                          int64
 10   icu_stay_type                   object
 11   icu_type                        object
 12   pre_icu_los_days                float64
 13   weight                          float64
```

3.3. Categorical Data Encoding

In [59]:
```
# Object type columns and corresponding number of unique values
print(pd.Series(data = [data[col].nunique() for col in cols_object], index
```

```
ethnicity                6
gender                   2
hospital_admit_source    15
icu_admit_source         5
icu_stay_type            3
icu_type                 8
apache_3j_bodysystem     11
apache_2_bodysystem      10
```

All 8 categorical features are nominal in nature, i.e. there is no notion of order in their realized values.

Label encoding

In [60]:
```
def label_encoder(df, cols):
    df_le = df.copy(deep = True)
    le = LabelEncoder()
    for col in cols:
        df_le[col] = le.fit_transform(df_le[col])
    return df_le
```

In [61]:
```
X_train.info(verbose=True)
```

```
<class 'pandas.core.frame.DataFrame'>
Index: 73370 entries, 52236 to 77555
Data columns (total 108 columns):
 #    Column                          Dtype
---   ------                          -----
 0    hospital_id                     int64
 1    age                             float64
 2    bmi                             float64
 3    elective_surgery                int64
 4    ethnicity                       object
 5    gender                          object
 6    height                          float64
 7    hospital_admit_source           object
 8    icu_admit_source                object
 9    icu_id                          int64
 10   icu_stay_type                   object
 11   icu_type                        object
 12   pre_icu_los_days                float64
 13   weight                          float64
```

Explanation of the arguments:

- **df:** The input dataset
- **cols:** List of columns that we want to encode

In [62]:
```
X_train_le = label_encoder(X_train, cols_object)
X_test_le = label_encoder(X_test, cols_object)
```

In [63]:
```
X_train.info(verbose=True)
```

```
<class 'pandas.core.frame.DataFrame'>
Index: 73370 entries, 52236 to 77555
Data columns (total 108 columns):
 #    Column                          Dtype
---   ------                          -----
 0    hospital_id                     int64
 1    age                             float64
 2    bmi                             float64
 3    elective_surgery                int64
 4    ethnicity                       object
 5    gender                          object
 6    height                          float64
 7    hospital_admit_source           object
 8    icu_admit_source                object
 9    icu_id                          int64
 10   icu_stay_type                   object
 11   icu_type                        object
 12   pre_icu_los_days                float64
 13   weight                          float64
```

For a categorical column with n distinct values, the label encoder maps the n distinct values to numerical values between 0 and $n-1$.

In [64]:
```
# Example
X_train_le[[col for col in X_train_le.columns if cols_object[1] in col]]
```

Out[64]:

	gender
52236	0
30240	1
40425	1
26444	1
26777	0
...	...
65294	1
16486	1
1888	0
64377	0
77555	1

73370 rows × 1 columns

One-hot encoding

In [65]:
```
# Function for one-hot encoding
def one_hot_encoder(df, cols, drop_first = False):
    cols = [col for col in cols if col in df.columns] # To ensure that 'col
    df_ohe = pd.get_dummies(df, columns = cols, drop_first = drop_first)
    return df_ohe
```

In [66]:
```
X_train.info(verbose=True)
```

```
<class 'pandas.core.frame.DataFrame'>
Index: 73370 entries, 52236 to 77555
Data columns (total 108 columns):
 #    Column                          Dtype
---   ------                          -----
 0    hospital_id                     int64
 1    age                             float64
 2    bmi                             float64
 3    elective_surgery                int64
 4    ethnicity                       object
 5    gender                          object
 6    height                          float64
 7    hospital_admit_source           object
 8    icu_admit_source                object
 9    icu_id                          int64
 10   icu_stay_type                   object
 11   icu_type                        object
 12   pre_icu_los_days                float64
 13   weight                          float64
```

Explanation of the arguments:

- **df:** The input dataset

- **cols:** List of columns that we want to encode

In [67]:
```
# One-hot encoding with drop_first = False
X_concat = pd.concat([X_train, X_test])
X_concat_ohe = one_hot_encoder(X_concat, cols_object, drop_first = False)
X_train_ohe = X_concat_ohe.loc[X_train.index]
X_test_ohe = X_concat_ohe.loc[X_test.index]
```

For a categorical column with k distinct values, the [one-hot (https://en.wikipedia.org/wiki/One-hot)](https://en.wikipedia.org/wiki/One-hot) encoder produces k new columns, one corresponding to each unique value. The original column is then dropped.

In [68]:
```
# Example
X_train_ohe[[col for col in X_train_ohe.columns if cols_object[1] in col]]
```

Out[68]:

	gender_F	gender_M
52236	True	False
30240	False	True
40425	False	True
26444	False	True
26777	True	False
...	...	...
65294	False	True
16486	False	True
1888	True	False
64377	True	False
77555	False	True

73370 rows × 2 columns

Note that `gender_F` and `gender_M` are related by `gender_F + gender_M = 1`. Hence we shall lose no information by dropping one of these columns. This can be done (for each feature) by changing `drop_first = False` to `drop_first = True`.

In [69]:
```
# One-hot encoding with drop_first = True
X_concat_ohe = one_hot_encoder(X_concat, cols_object, drop_first = True)
X_train_ohe = X_concat_ohe.loc[X_train.index]
X_test_ohe = X_concat_ohe.loc[X_test.index]
```

Now the one-hot encoder produces $k-1$ new columns for a categorical column with k distinct values, by dropping the first column. As before, the original column is also dropped.

In [70]:
```
# Example
X_train_ohe[[col for col in X_train_ohe.columns if cols_object[1] in col]]
```

Out[70]:

	gender_M
52236	False
30240	True
40425	True
26444	True
26777	False
...	...
65294	True
16486	True
1888	False
64377	False
77555	True

73370 rows × 1 columns

To have a column corresponding to `nan` values (if they are present in the original column), one must set the `dummy_na` parameter of the `get_dummies` function to be `True` (which is by default `False`).

In [71]:
```
# Replacing original predictors with encoded predictors
X_train = X_train_ohe
X_test = X_test_ohe
```

3.4. Normalization

In [72]:
```
# Min-max normalization of predictors in the training set
for col in X_train.columns:
    if X_train[col].dtypes == 'int64' or X_train[col].dtypes == 'float64':
        if X_train[col].nunique() > 1: # Checking if the column is non-cons
            X_train[col] = (X_train[col] - X_train[col].min()) / (X_train[c
X_train
```

Out[72]:

	hospital_id	age	bmi	elective_surgery	height	icu_id	pre_icu_los_days
52236	0.648515	0.698630	0.306612	1.0	0.260319	0.552663	0.13735
30240	0.425743	0.178082	0.308950	0.0	0.715876	0.376331	0.13569
40425	0.782178	0.794521	0.214400	1.0	0.520637	0.459172	0.18022
26444	0.717822	0.232877	0.096394	0.0	0.664497	0.278107	0.13574
26777	0.787129	0.301370	0.125461	0.0	0.424730	0.391716	0.13562
...	...	...	...	...	...	...	...
65294	0.202970	0.438356	0.209678	0.0	0.561740	0.662722	0.13563
16486	0.014851	0.986301	0.177802	0.0	0.825484	0.211834	0.16111
1888	0.400990	0.479452	0.543196	1.0	0.347662	0.015385	0.13812
64377	0.084158	0.438356	0.195987	0.0	0.613119	0.675740	0.13555
77555	0.960396	0.863014	0.087310	0.0	0.565165	0.886391	0.13701

73370 rows × 152 columns

In [73]:
```
X_train.info(verbose=True)
```

```
<class 'pandas.core.frame.DataFrame'>
Index: 73370 entries, 52236 to 77555
Data columns (total 152 columns):
 #    Column                                  Dtype
---   ------                                  -----
 0    hospital_id                             float64
 1    age                                     float64
 2    bmi                                     float64
 3    elective_surgery                        float64
 4    height                                  float64
 5    icu_id                                  float64
 6    pre_icu_los_days                        float64
 7    weight                                  float64
 8    apache_2_diagnosis                      float64
 9    apache_3j_diagnosis                     float64
 10   apache_post_operative                   float64
 11   arf_apache                              float64
 12   bun_apache                              float64
 13   creatinine_apache                       float64
```

In [74]:
```
# Min-max normalization of predictors in the test set
for col in X_test.columns:
    if X_test[col].dtypes == 'int64' or X_test[col].dtypes == 'float64': #
        if X_test[col].nunique() > 1: # Checking if the column is non-const
            X_test[col] = (X_test[col] - X_test[col].min()) / (X_test[col].m
X_test
```

Out[74]:

	hospital_id	age	bmi	elective_surgery	height	icu_id	pre_icu_los_days
48467	0.094059	0.753425	0.243972	0.0	0.520637	0.499408	0.180556
65356	0.084158	0.958904	0.259324	0.0	0.561740	0.675740	0.18050
16224	0.564356	0.876712	0.077817	0.0	0.477822	0.266272	0.180592
44085	0.910891	0.698630	0.334880	0.0	0.825484	0.492308	0.29794
79745	0.500000	0.602740	0.240685	0.0	0.607981	0.847337	0.18233
...	...	...	...	...	...	...	..
51289	0.648515	0.589041	0.123187	1.0	0.435006	0.552663	0.18573
37193	0.336634	0.835616	0.311336	0.0	0.544614	0.436686	0.23310
83377	0.222772	0.287671	0.380793	0.0	0.253468	0.887574	0.18081
89064	0.920792	0.808219	0.220514	0.0	0.347662	0.898225	0.18024
411	0.574257	0.287671	0.233699	0.0	0.738140	0.020118	0.18025

18343 rows × 152 columns

In [75]:
```
X_train.info(verbose=True)
```

```
<class 'pandas.core.frame.DataFrame'>
Index: 73370 entries, 52236 to 77555
Data columns (total 152 columns):
 #    Column                                  Dtype
---   ------                                  -----
 0    hospital_id                             float64
 1    age                                     float64
 2    bmi                                     float64
 3    elective_surgery                        float64
 4    height                                  float64
 5    icu_id                                  float64
 6    pre_icu_los_days                        float64
 7    weight                                  float64
 8    apache_2_diagnosis                      float64
 9    apache_3j_diagnosis                     float64
 10   apache_post_operative                   float64
 11   arf_apache                              float64
 12   bun_apache                              float64
 13   creatinine_apache                       float64
```

In [80]:
```
X_train = X_train.astype(float)
X_test = X_test.astype(float)
```

In [81]:
```
X_test.info(verbose=True)
```

```
<class 'pandas.core.frame.DataFrame'>
Index: 18343 entries, 48467 to 411
Data columns (total 152 columns):
 #    Column                                  Dtype
---   ------                                  -----
 0    hospital_id                             float64
 1    age                                     float64
 2    bmi                                     float64
 3    elective_surgery                        float64
 4    height                                  float64
 5    icu_id                                  float64
 6    pre_icu_los_days                        float64
 7    weight                                  float64
 8    apache_2_diagnosis                      float64
 9    apache_3j_diagnosis                     float64
 10   apache_post_operative                   float64
 11   arf_apache                              float64
 12   bun_apache                              float64
 13   creatinine_apache                       float64
```

4. Baseline Neural Network

In [82]:
```
# Adding layers to sequential model
model = Sequential()
model.add(Dense(16, input_dim = len(X_train.columns), activation = 'relu'))
model.add(Dense(12, activation = 'relu'))
model.add(Dense(8, activation = 'relu'))
model.add(Dense(4, activation = 'relu'))
model.add(Dense(1, activation = 'sigmoid'))
model.summary()
```

```
Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 dense (Dense)               (None, 16)                2448

 dense_1 (Dense)             (None, 12)                204

 dense_2 (Dense)             (None, 8)                 104

 dense_3 (Dense)             (None, 4)                 36

 dense_4 (Dense)             (None, 1)                 5

=================================================================
Total params: 2797 (10.93 KB)
Trainable params: 2797 (10.93 KB)
Non-trainable params: 0 (0.00 Byte)
_________________________________________________________________
```

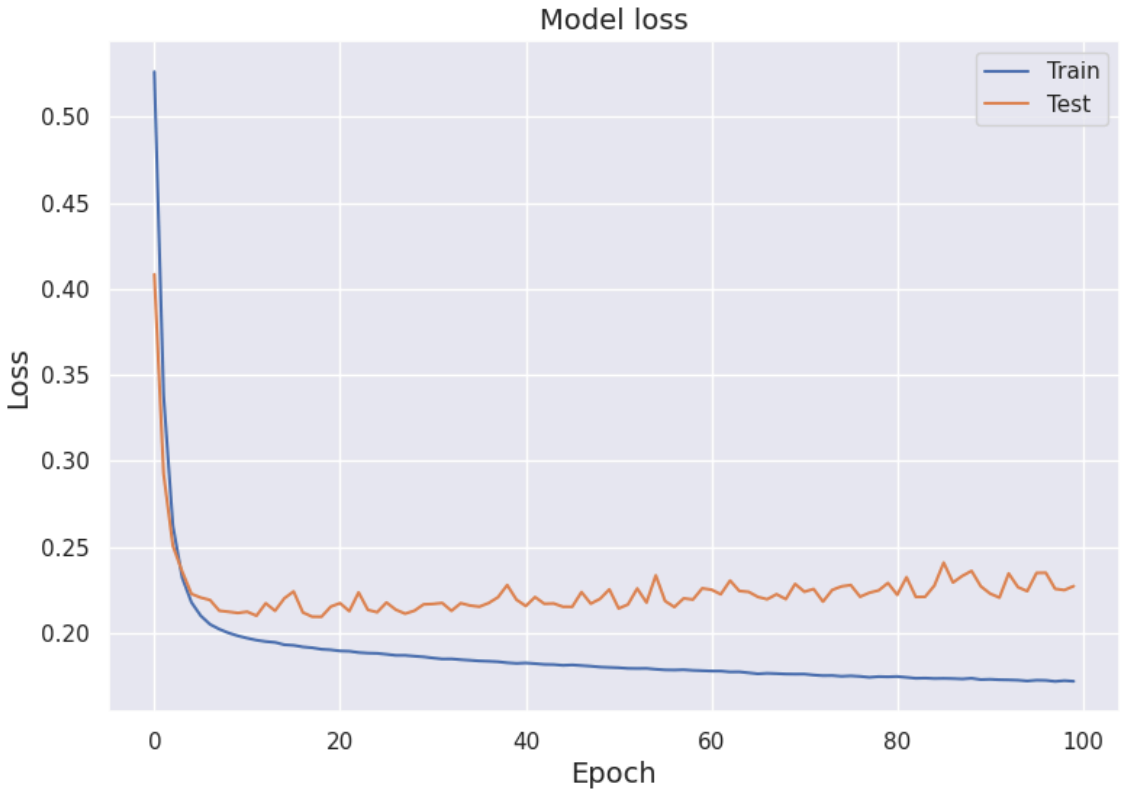
In [83]:
```
# Specifying loss function and optimizer
model.compile(loss = 'binary_crossentropy', optimizer = 'adam', metrics = [
```

In [84]:
```
# Training the model
history = model.fit(X_train, y_train, validation_data = (X_test, y_test), e
```

```
Epoch 1/100
1147/1147 [==============================] - 8s 6ms/step - loss: 0.5262
- accuracy: 0.8998 - val_loss: 0.4085 - val_accuracy: 0.9127
Epoch 2/100
1147/1147 [==============================] - 6s 5ms/step - loss: 0.3381
- accuracy: 0.9200 - val_loss: 0.2926 - val_accuracy: 0.9212
Epoch 3/100
1147/1147 [==============================] - 3s 3ms/step - loss: 0.2625
- accuracy: 0.9248 - val_loss: 0.2507 - val_accuracy: 0.9221
Epoch 4/100
1147/1147 [==============================] - 5s 4ms/step - loss: 0.2322
- accuracy: 0.9261 - val_loss: 0.2358 - val_accuracy: 0.9190
Epoch 5/100
1147/1147 [==============================] - 3s 3ms/step - loss: 0.2179
- accuracy: 0.9264 - val_loss: 0.2228 - val_accuracy: 0.9225
Epoch 6/100
1147/1147 [==============================] - 3s 3ms/step - loss: 0.2101
- accuracy: 0.9263 - val_loss: 0.2206 - val_accuracy: 0.9216
Epoch 7/100
```

In [85]:
```
# Visualization of model accuracy
model_accuracy = pd.DataFrame()
model_accuracy['accuracy'] = history.history['accuracy']
model_accuracy['val_accuracy'] = history.history['val_accuracy']

plt.figure(figsize = (9, 6))
sns.lineplot(data = model_accuracy['accuracy'], label = 'Train')
sns.lineplot(data = model_accuracy['val_accuracy'], label = 'Test')
plt.title('Model accuracy', fontsize = 14)
plt.ylabel('Accuracy', fontsize = 14)
plt.xlabel('Epoch', fontsize = 14)
plt.legend()
plt.show()
```

In [86]:
```
# Visualization of model loss
model_loss = pd.DataFrame()
model_loss['loss'] = history.history['loss']
model_loss['val_loss'] = history.history['val_loss']

plt.figure(figsize = (9, 6))
sns.lineplot(data = model_loss['loss'], label = 'Train')
sns.lineplot(data = model_loss['val_loss'], label = 'Test')
plt.title('Model loss', fontsize = 14)
plt.ylabel('Loss', fontsize = 14)
plt.xlabel('Epoch', fontsize = 14)
plt.legend()
plt.show()
```

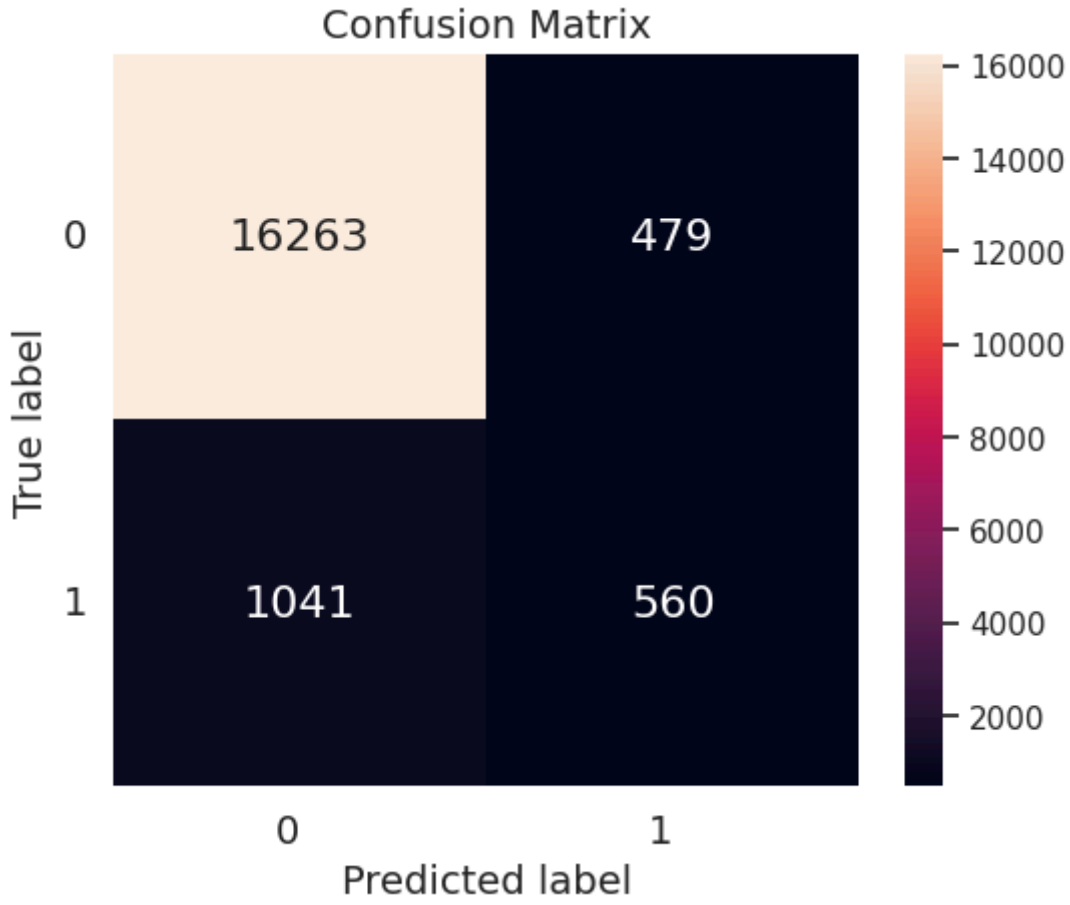


In [87]:
```
# Prediction on test set
pred = model.predict(X_test)
threshold = 0.5
y_pred = [0 if pred[i][0] < threshold else 1 for i in range(len(pred))]
```

```
574/574 [==============================] - 1s 1ms/step
```

In [89]:
```
# Function to compute and visualize confusion matrix
def confusion_mat(y_pred, y_test):
    class_names = [0, 1]
    tick_marks_y = [0.5, 1.5]
    tick_marks_x = [0.5, 1.5]
    confusion_matrix = metrics.confusion_matrix(y_test, y_pred)
    confusion_matrix_df = pd.DataFrame(confusion_matrix, range(2), range(2)
    plt.figure(figsize = (6, 4.75))
    plt.title("Confusion Matrix", fontsize = 14)
    hm = sns.heatmap(confusion_matrix_df, annot = True, annot_kws = {"size"
    hm.set_xlabel("Predicted label", fontsize = 14)
    hm.set_ylabel("True label", fontsize = 14)
    hm.set_xticklabels(class_names, fontdict = {'fontsize': 14}, rotation =
    hm.set_yticklabels(class_names, fontdict = {'fontsize': 14}, rotation =
    plt.grid(False)
    plt.show()
```

In [90]:
```
# Confusion matrix
confusion_mat(y_pred, y_test)
```



In [91]:
```
# Evaluation metrics
acc = accuracy_score(y_test, y_pred)
roc_auc = roc_auc_score(y_test, y_pred)
precision = precision_score(y_test, y_pred)
recall = recall_score(y_test, y_pred)
f1 = f1_score(y_test, y_pred)

print(pd.Series({"Accuracy": acc,
                 "ROC-AUC": roc_auc,
                 "Precision": precision,
                 "Recall": recall,
                 "F1-score": f1}).to_string())
```

```
Accuracy     0.917135
ROC-AUC      0.660585
Precision    0.538980
Recall       0.349781
F1-score     0.424242
```

5. Hyperparameter Tuning

In [92]:
```
# Building the model
def model_builder(ht):
    model = Sequential()
    model.add(keras.layers.Flatten(input_shape = (X_train.shape[1],)))

    # Tuning the number of units in the first Dense layer
    ht_units = ht.Int('units', min_value = 32, max_value = 512, step = 32)
    model.add(keras.layers.Dense(units = ht_units, activation = 'relu'))
    model.add(keras.layers.Dense(12, activation = 'relu'))
    model.add(keras.layers.Dense(8, activation = 'relu'))
    model.add(keras.layers.Dense(4, activation = 'relu'))
    model.add(keras.layers.Dense(1, activation = 'sigmoid'))

    # Tuning the learning rate for the optimizer
    ht_learning_rate = ht.Choice('learning_rate', values = [0.01, 0.001, 0.

    model.compile(loss = 'binary_crossentropy', optimizer = Adam(learning_r

    return model
```

In [93]:
```
# Making the tuner
tuner = kt.Hyperband(model_builder,
                     objective = 'val_accuracy',
                     max_epochs = 10,
                     factor = 3,
                     directory = 'dir_2')
```

In [94]:
```
# Early stopping
stop_early = tf.keras.callbacks.EarlyStopping(monitor = 'val_loss', patienc
```

In [95]:
```
# Implementing the tuner
tuner.search(X_train, y_train, epochs = 50, validation_split = 0.2, callbac

# Get the optimal hyperparameters
best_hparams = tuner.get_best_hyperparameters(num_trials = 1)[0]

print("-------- The hyperparameter search is complete --------")
print(" ")
print(pd.Series({"Optimal number of units in the first densely-connected la
                 "Optimal learning rate for the optimizer": best_hparams.ge
```

```
Trial 30 Complete [00h 01m 07s]
val_accuracy: 0.9300122857093811

Best val_accuracy So Far: 0.9304211735725403
Total elapsed time: 00h 20m 31s
-------- The hyperparameter search is complete --------

Optimal number of units in the first densely-connected layer    416.000
Optimal learning rate for the optimizer                           0.001
```

In [96]:
```
# Building the model with optimal hyperparameters
model = tuner.hypermodel.build(best_hparams)

# Training the model
history = model.fit(X_train, y_train, epochs = 50, validation_split = 0.2)

# Validation accuracy
val_accuracy_optimal = history.history['val_accuracy']

# Computing best epoch in terms of maximum validation accuracy
best_epoch = val_accuracy_optimal.index(max(val_accuracy_optimal)) + 1
print(" ")
print(pd.Series({"Best epoch": (best_epoch)}).to_string())
```

```
Epoch 1/50
1835/1835 [==============================] - 10s 4ms/step - loss: 0.308
2 - accuracy: 0.9169 - val_loss: 0.2117 - val_accuracy: 0.9222
Epoch 2/50
1835/1835 [==============================] - 9s 5ms/step - loss: 0.2093
- accuracy: 0.9230 - val_loss: 0.2015 - val_accuracy: 0.9267
Epoch 3/50
1835/1835 [==============================] - 8s 4ms/step - loss: 0.2045
- accuracy: 0.9246 - val_loss: 0.2007 - val_accuracy: 0.9278
Epoch 4/50
1835/1835 [==============================] - 9s 5ms/step - loss: 0.2011
- accuracy: 0.9262 - val_loss: 0.1960 - val_accuracy: 0.9290
Epoch 5/50
1835/1835 [==============================] - 7s 4ms/step - loss: 0.1987
- accuracy: 0.9274 - val_loss: 0.1982 - val_accuracy: 0.9268
Epoch 6/50
1835/1835 [==============================] - 10s 5ms/step - loss: 0.195
5 - accuracy: 0.9280 - val_loss: 0.1956 - val_accuracy: 0.9290
Epoch 7/50
```

In [ ]:
```
# Re-instantiating the model
model_tuned = tuner.hypermodel.build(best_hparams)

# Re-training the hypermodel with the optimal number of epochs
model_tuned.fit(X_train, y_train, epochs = best_epoch, validation_split = 0
```

In [98]:
```
# Evaluation on the test set
eval_tuned = model_tuned.evaluate(X_test, y_test)
print(" ")
print(pd.Series({"Test loss": eval_tuned[0],
                 "Test accuracy": eval_tuned[1]}).to_string())
```

```
574/574 [==============================] - 1s 2ms/step - loss: 0.2119 - ac
curacy: 0.9192

Test loss        0.211931
Test accuracy    0.919206
```

In [ ]:
```
# Confusion matrix
pred_tuned = model_tuned.predict(X_test)
threshold = 0.5
y_pred_tuned = [0 if pred_tuned[i][0] < threshold else 1 for i in range(len
confusion_mat(y_pred_tuned, y_test)
```

In [100]:
```
# Evaluation metrics
acc = accuracy_score(y_test, y_pred_tuned)
roc_auc = roc_auc_score(y_test, y_pred_tuned)
precision = precision_score(y_test, y_pred_tuned)
recall = recall_score(y_test, y_pred_tuned)
f1 = f1_score(y_test, y_pred_tuned)

print(pd.Series({"Accuracy": acc,
                 "ROC-AUC": roc_auc,
                 "Precision": precision,
                 "Recall": recall,
                 "F1-score": f1}).to_string())
```

```
Accuracy     0.919206
ROC-AUC      0.666522
Precision    0.557488
Recall       0.360400
F1-score     0.437785
Accuracy     0.919206
ROC-AUC      0.666522
Precision    0.557488
Recall       0.360400
F1-score     0.437785
```

Saving and loading the model

In [101]:
```
# Saving the model
model_tuned.save('model_tuned.h5')
```

In [102]:
```
# Loading the model
model_tuned_loaded = load_model('model_tuned.h5')
```

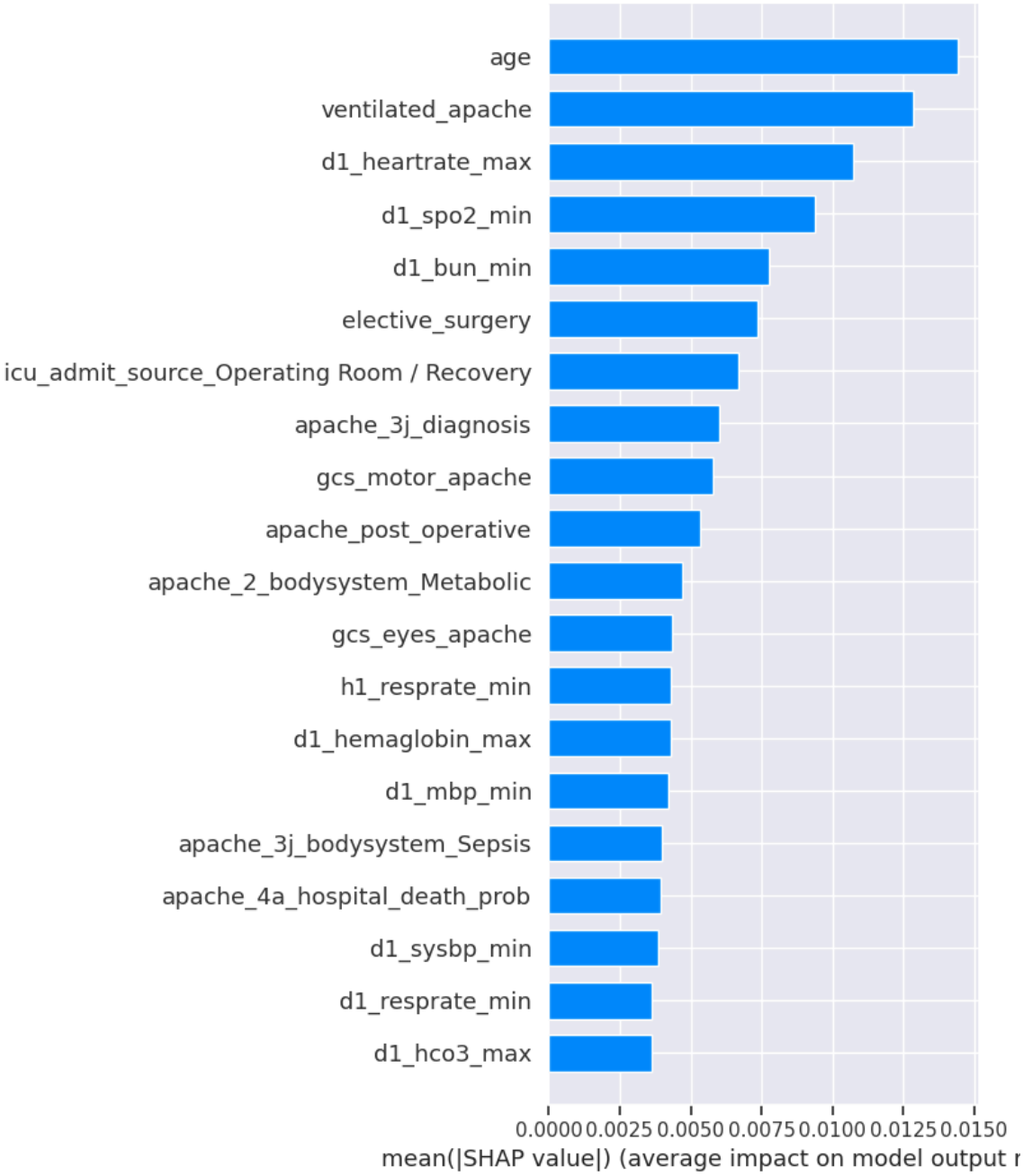
6. Explainable AI

In [103]:
```
# Loading JavaScript library
shap.initjs()

# Sampling from test data predictors
X_test_sample = X_test.sample(50)

# Predicted values corresponding to the sample
pred_tuned_sample = np.array(pd.Series(data = pred_tuned.flatten(), index =
y_pred_tuned_sample = np.array(pd.Series(data = y_pred_tuned, index = X_tes

# Wrapper function
def wrap(X):
    return model_tuned.predict(X).flatten()

# Explainer
explainer = shap.KernelExplainer(wrap, X_test_sample)

# Computing SHAP values based on the sample
shap_values = explainer.shap_values(X_test_sample, nsamples = 500)
```

```
2/2 [==============================] - 0s 9ms/step

  0%|          | 0/50 [00:00<?, ?it/s]

1/1 [==============================] - 0s 22ms/step
782/782 [==============================] - 2s 2ms/step
1/1 [==============================] - 0s 22ms/step
782/782 [==============================] - 2s 2ms/step
1/1 [==============================] - 0s 30ms/step
782/782 [==============================] - 2s 3ms/step
1/1 [==============================] - 0s 21ms/step
782/782 [==============================] - 2s 2ms/step
1/1 [==============================] - 0s 21ms/step
782/782 [==============================] - 2s 2ms/step
1/1 [==============================] - 0s 21ms/step
782/782 [==============================] - 2s 3ms/step
1/1 [==============================] - 0s 107ms/step
782/782 [==============================] - 5s 6ms/step
```

Global interpretation

In [104]:
```
# Summary plot
shap.summary_plot(shap_values = shap_values, features = X_test_sample, plot
```



Local interpretation

Explaining single prediction

In [105]:
```
# Force plot
shap.initjs()
row = math.floor(3*len(X_test_sample)/4)
print(pd.Series({"Predicted value": pred_tuned_sample[row]}).to_string())
shap_values_row = explainer.shap_values(X_test_sample.iloc[row, :], nsample
shap.force_plot(base_value = explainer.expected_value,
                shap_values = shap_values_row,
                features = X_test_sample.iloc[row, :],
                feature_names = X_test_sample.columns)
```

```
Predicted value    0.102738
1/1 [==============================] - 0s 24ms/step
782/782 [==============================] - 1s 2ms/step
```

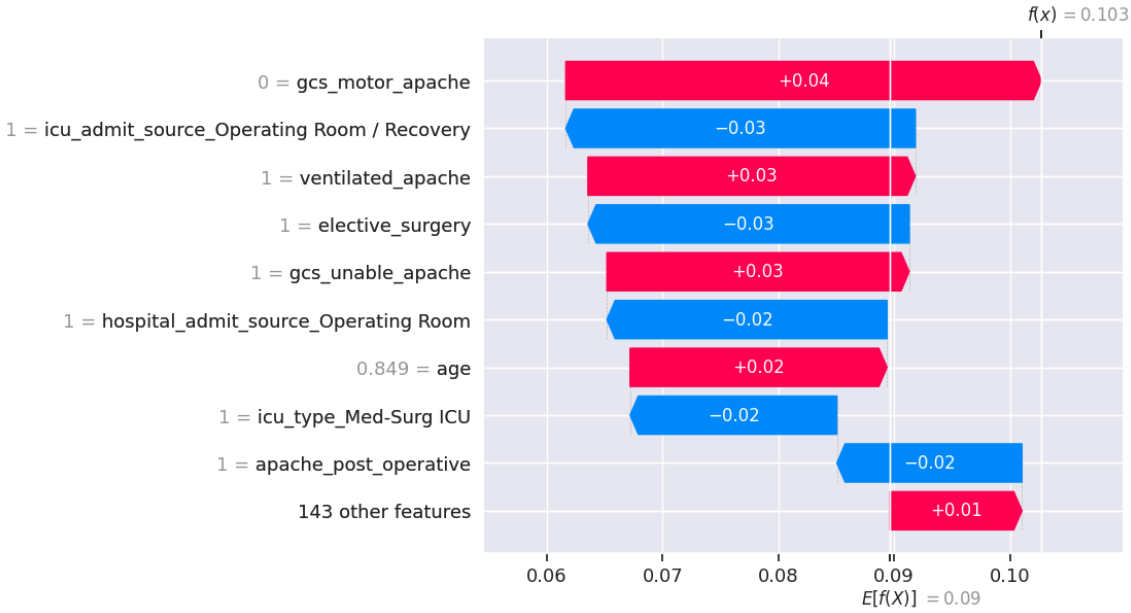
Out[105]:

Visualization omitted, Javascript library not loaded!

Have you run `initjs()` in this notebook? If this notebook was from another user you must also trust this notebook (File -> Trust notebook). If you are viewing this notebook on github the Javascript has been stripped for security. If you are using JupyterLab this error is because a JupyterLab extension has not yet been written.

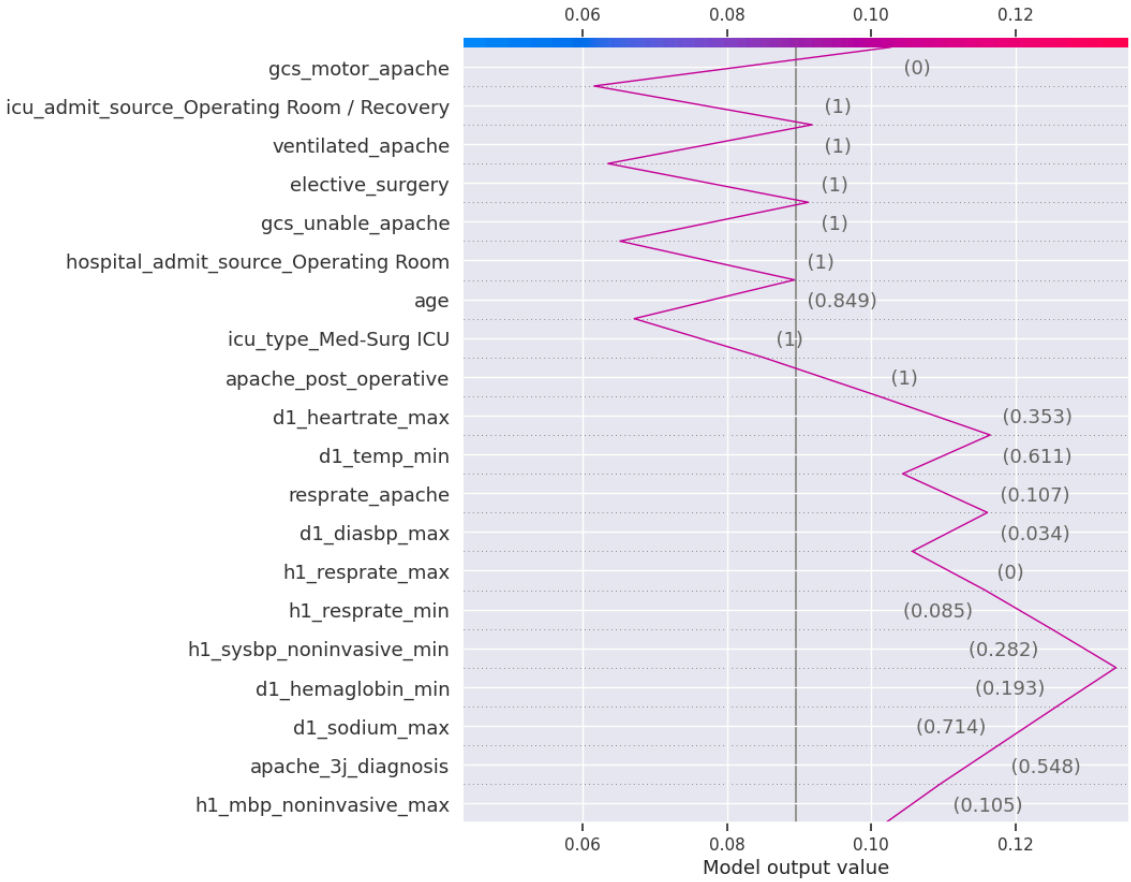
In [106]:
```
# Waterfall plot
print(pd.Series({"Predicted value": pred_tuned_sample[row]}).to_string())
shap.waterfall_plot(shap.Explanation(values = shap_values_row,
                                     base_values = explainer.expected_value
                                     data = X_test_sample.iloc[row, :],
                                     feature_names = X_test_sample.columns.
```

```
Predicted value    0.102738
```



In [107]:
```
# Decision plot
print(pd.Series({"Predicted value": pred_tuned_sample[row]}).to_string())
shap.decision_plot(base_value = explainer.expected_value,
                   shap_values = shap_values_row,
                   features = X_test_sample.iloc[row, :],
                   feature_names = X_test_sample.columns.tolist())
```

```
Predicted value    0.102738
```



Explaining multiple predictions

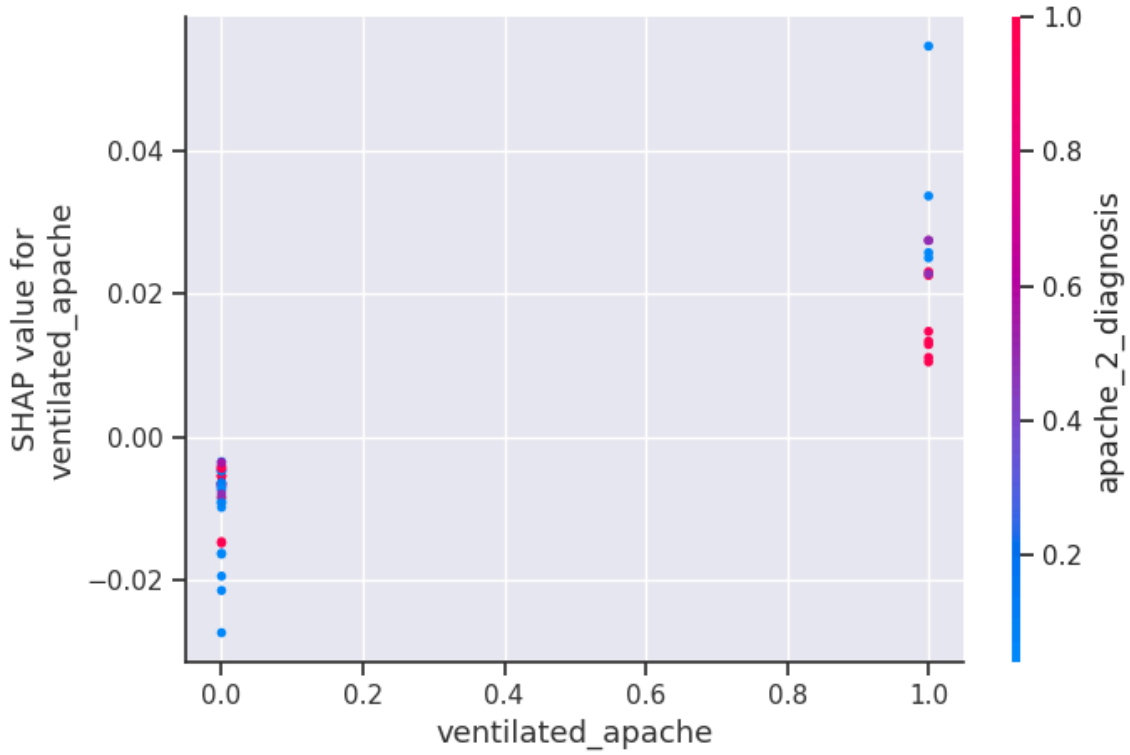
In [108]:
```
# Force plot
shap.initjs()
shap.force_plot(base_value = explainer.expected_value,
                shap_values = shap_values,
                features = X_test_sample,
                feature_names = X_test_sample.columns)
```

Out[108]:

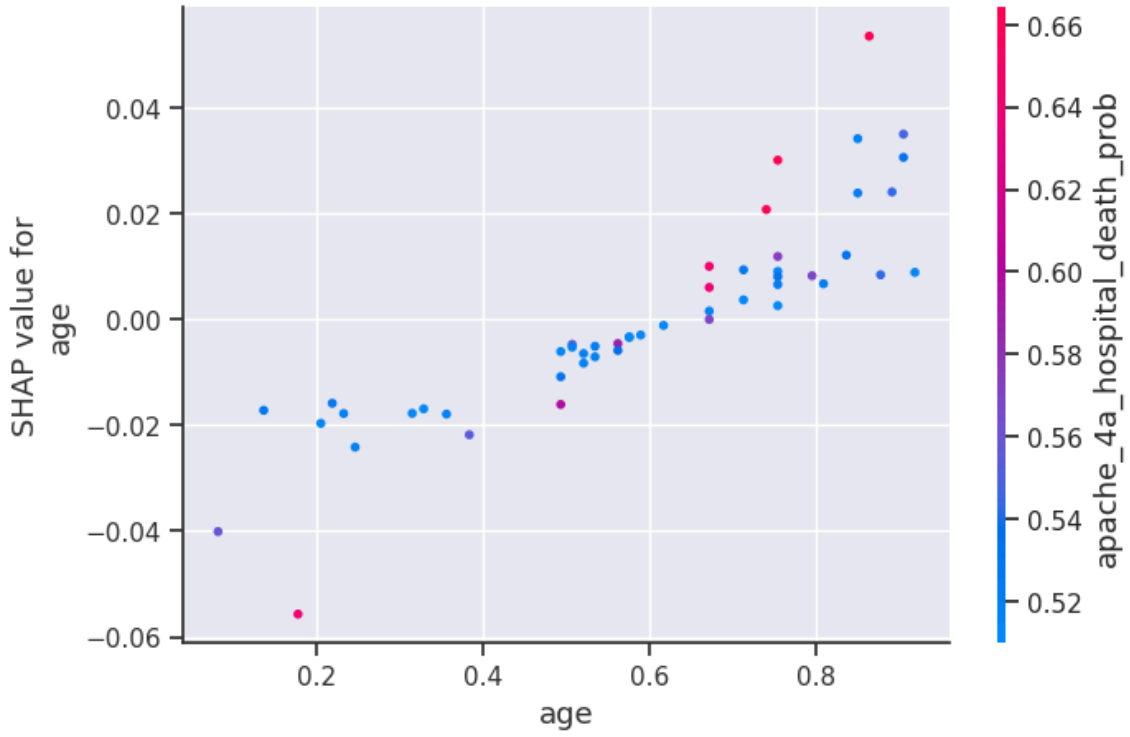
Visualization omitted, Javascript library not loaded!

Have you run `initjs()` in this notebook? If this notebook was from another user you must also trust this notebook (File -> Trust notebook). If you are viewing this notebook on github the Javascript has been stripped for security. If you are using JupyterLab this error is because a JupyterLab extension has not yet been written.

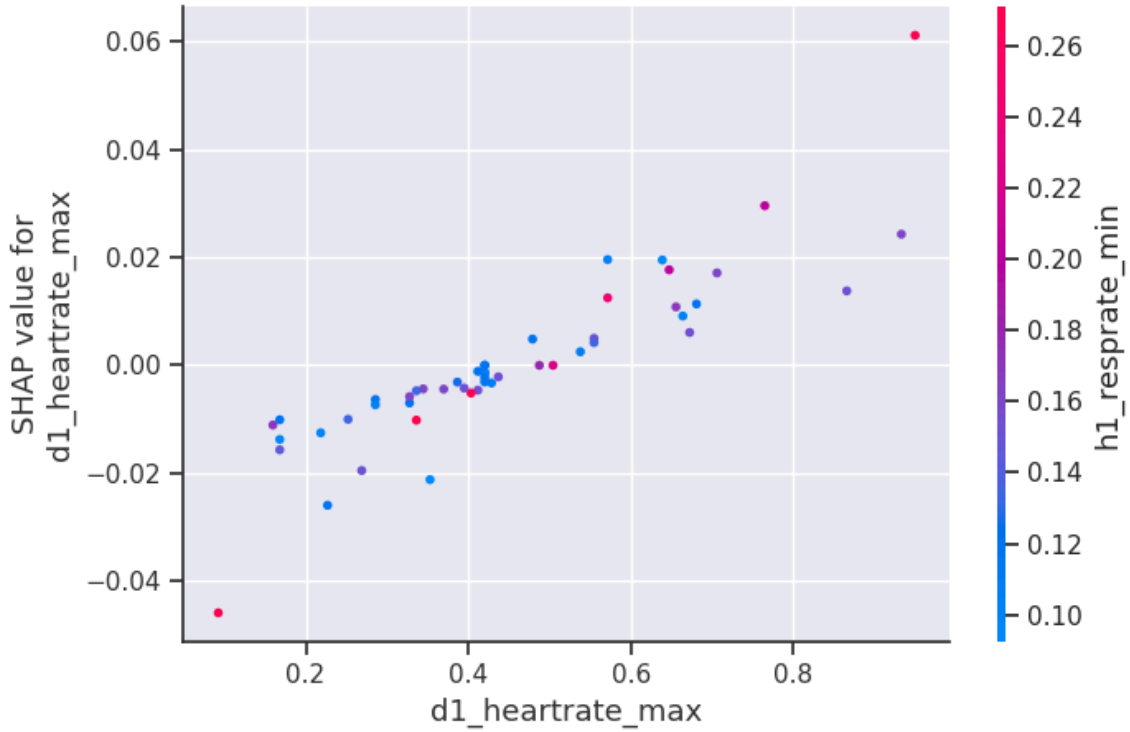
In [109]:
```
# Dependence plot for ventilated_apache
shap.dependence_plot(ind = 'ventilated_apache', shap_values = shap_values,
```



In [110]:
```
# Dependence plot for age
shap.dependence_plot(ind = 'age', shap_values = shap_values, features = X_t
```



In [111]:
```
# Dependence plot for age
shap.dependence_plot(ind = 'd1_heartrate_max', shap_values = shap_values, f
```



In [112]:
```
# Decision plot
print("Predicted values")
print(" ")
rows = [i*math.floor(len(X_test_sample)/6) for i in range(6)]
print(np.array(pd.Series(pred_tuned_sample)[rows]))
print(" ")
shap.decision_plot(base_value = explainer.expected_value,
                   shap_values = shap_values[rows],
                   features = X_test_sample.iloc[rows, :],
                   feature_names = X_test_sample.columns.tolist())
```

```
Predicted values

[0.20547847 0.25853458 0.00114079 0.01458947 0.00921752 0.00199346]
```