D214 Capstone

Task 2: Analytics Report

Petra I Bier

Western Governors University

Capstone Project for WGU MSDA program

April 2, 2025

Dr. Daniel Smith, PhD

Table of Contents

A: Research Question

The context of this analysis is the common usage of SpO2 in hospitals and clinics. SpO2 is a convenient, non-invasive measurement of the blood's oxygen levels. A major benefit of SpO2 reading is that it can be done continuously at a low cost. This stands in contrast to the gold standard for blood oxygen saturation measurement, SaO2, which is collected via arterial blood gas (ABG). This is an invasive procedure where blood is drawn from an artery with inherent risks, such as bleeding, infection, and blood clots. Since it is not a continuous measurement, multiple draws are often required.

Research has demonstrated that SpO2 does not always correlate accurately with SaO2, especially at lower levels (Sjoding et al., 2023). Hidden hypoxemia, which is when SpO2 is > 88% yet the SaO2 is < 88%, may be present and remain undetected if solely relying on SpO2 readings, leading to inaccurate diagnosis and delayed treatment of hypoxemia. This, in turn, can lead to extended hospitalizations and poor patient outcomes.

Building a statistical model to predict the SaO2 value that uses only non-invasive measurement could lead to improved hypoxemia diagnosis while reducing the risks associated with arterial access. Venous blood draws are a common method used to obtain lab measurements and are frequently done in conjunction when starting intravenous (IV) treatments. The skills and risks associated with venous access are significantly lower than arterial draws and measurements, allowing smaller hospitals and clinics the ability to use these methods easily.

The goal of this analysis is to answer the question: Can a random forest regression (RFR) model predict the difference between a pulse oximeter (SpO2) and arterial oxygen saturation (SaO2) reading using the given data set?

For this analysis the null and alternative hypothesis are as follows:

> Null hypothesis- A random forest model cannot detect the difference between the SpO2 and the SaO2 with an RMSE of less than 3 percentage points.

> Alternate Hypothesis- A random forest model can detect the difference between the SpO2 and the SaO2 with an RMSE of less than 3 percentage points.

A random forest regression model will be used to analyze non-linear relationships from the chosen predictor variables to predict SpO2 accuracy in alignment with the FDA error requirements. A predictive model that can predict the SpO2 – SaO2 gap to determine the accuracy of the SpO2 measurement will be created and evaluated. The FDA requires that pulse oximetry is accurate withing 2-3% of SaO2 levels only two-thrids of the time (Wong et al., 2021). This value aligns with the hypotheses testing of 3% RMSE, which is equivalent to three percentage points of SpO2.

B: Data Collection

The data used for this analysis was sourced from the Physionet BOLD data set (Goldberger et al., 2000; Matos et al., 2023a). The data was accessed after completing the required training and data usage agreement (DUA). The training included topics of ethics, privacy principals and HIPPA compliance. While Physionet does have data that can be considered identifiable, this particular data set was de-identified for public usage. It was also deemed exempt from human subject research. The data was collected

from three Electronic Health Record (EHR) databases (MIMIC-III, MIMIC-IV, eICU-CRD). These are large, publicly available data sets that have retrospectively gathered data from ICUs at large hospitals, and all patient information had been de-identified within each database. The eICU Collaborative Database is data collected from EHR data from 200,000 admissions between 2014–2015. The Medical Information Mart for Intensive Care (MIMIC)-III database contains de-identified information collected from 40,000 EHR records from the Beth Israel Deaconess Medical Center (BIDMC) between 2001 and 2012. MIMIC-IV data was also sourced from BIDMC from 2008–2009. The overlap between the dataset was addressed and delt with through tracking and matching of the original patient identifiers, which were then dropped for unique identifiers to this data set. The blood-gas and oximetry linked dataset (BOLD) found SpO2 readings that were aligned with SaO2 readings taken within a five-minute time frame. Any oxygen saturation reading not within the 70–100% range was not collected for this dataset. Variables across the three datasets were standardized, with only those available across all databases being used to reduce the number of null values. After synchronization of all variables was done across the three sets of data, they were then merged into the single data frame. All physiologic measurements were recorded in time in relation to the time the ABG was drawn. These times were recorded as "delta_" values. BMI values were calculated based on the admission height and weight entered. The Sequential Organ Failure (SOFA) score was also calculated from the newly combined data set. The original data set had 49,093 rows and 142 variables. It contained approximately 1,719,765 NaN values out of 6,971,206 total values (24.67%). It was presented in a CSV file with an accompanying PDF for the data dictionary. A link to the data set can be found here:

https://physionet.org/content/blood-gas-oximetry/1.0/

An advantage of using this data set was the ease of access to a large volume of information. A disadvantage was that though the data set was extensive, the user has little control of the variables chosen. Without direct control of what the variables were, the analyst must base the analysis on the data at hand. Several important variables were not included in this data set. To mitigate this problem, feature engineering was used to create several variables from the given data.

Columns created through feature engineering:

- Gap: Target variable, SpO2 - SaO2
- Oxygen carrying capacity (Grippi, 2020): Hemoglobin x 1.34
- BUN/ Creatinine ratio: BUN/ Creatinine
- Predicted body weight (PBW)(ARDSNet Protocol Card, 2025):
  - Male: 50 + 0.91 × (Height – 152.4)
  - Female: 45.5 + 0.91 × (Height – 152.4)

The final variables used in the analysis included:

- Demographics:

    - Age, Sex, Ethnicity
- Vital signs:
    - SpO2, heart rate, systolic blood pressure (SBP), mean blood pressure (MBP)
    - Respirations, temperature
    - Weight, height, body mass index (BMI)
- Laboratory values:
    - Blood cell metrics: hemoglobin, hematocrit, MCH, MCHC, MCV, RBC, RDW, white blood cells
    - Electrolytes: sodium, potassium, bicarbonate, anion gap, calcium
    - Metabolic/ Kidney: BUN, creatinine, glucose, lactate, albumin
    - Liver function: ALP, ALT, AST

A challenge during data cleaning was the handling of outliers. While some extreme values might represent errors, others capture rare but clinically significant cases. This analysis aimed to identify these potentially meaningful events. Distinguishing between erroneous outliers and non-error values required careful attention. A considerable amount of time was spent evaluating the values by hand, as well as researching and applying domain knowledge when setting outlier limits.

After cleaning and feature engineering, the final data set contained 49,093 rows and 38 columns. The sparsity of the cleaned data frame was 19.15%.

C: Data Extraction and Preparation

> **Note**: All code is presented in this notebook and is used as an equivalent to the requested screenshots for proof of work. Explanations have also been provided throughout the notebook to further explain the steps taken in the handling of the data prior to the building of the model. These can be seen interspersed between different code cells.

The programming language and environment for the RFR analysis will be Python using a Jupyter Notebook in Pycharm. Python was chosen for its ease of use and the large number of open-source libraries. It is easily scalable for large and small projects, and is strong in general software engineering, allowing analysis tools to be easily integrated into applications and websites. A disadvantage to using python is it can have longer run times since it is an interpreted language rather than a compiled language (McKinney, 2022).

Jupyter Notebooks offers advantages through its cell-by-cell execution of code. Visualizations appear directly below the executed code, which facilitates the exploratory phase of analysis. A downfall is that a non-linear workflow can be created, leading to errors during execution of the code and inconsistencies in the analysis process.

PyCharm is an integrated development environment (IDE) with strong support for virtual

environments and package management. Virtual environments allow for the use of different versions of packages without altering the native python package on a device. A disadvantage to PyCharm is the steeper learning curve, as it is not as intuitive as other IDEs. It also does not have improved text capabilities, having only Markdown capabilities for text cells.

The data was extracted and prepared using several different packages and libraries:

- Pandas
- NumPy
- Matplotlib
- Seaborn
- SciPy Stats
- SciKit- Learn
- SMOGN

C1: Data Handling and Exploration

The Pandas package was used to import and create the data frame. This library is known for its flexibility in creating and manipulating data structures. Unfortunately, it can be memory intensive when dealing with large data sets.

- import pandas as pd

NumPy can do computations on entire arrays, which often negates the need to code loops for calculations. The SciKit-Learn package is built upon NumPy due to its computational functionality, which was the main use in this RFR analysis. A disadvantage of this package is the limited functionality in comparison to Pandas for advanced data analysis.

- import numpy as np

For visual data exploration, the Matplotlib and Seaborn libraries were used. Matplotlib has a high level of customization but can become very verbose when creating more complex figures. Seaborn is built on top of Matplotlib, and does well at creating appealing visualizations, but can be challenging to customize. The two libraries are often used in conjunction with one another.

- import matplotlib.pyplot as plt
- import seaborn as sns

In [1]:
```
# Set up of notebook
import pandas as pd # used to create the dataframe
import numpy as np  #Required dependency for scikit.learn to run
import matplotlib.pyplot as plt # to visualize the data
import seaborn as sns #Used to visualize the data
from sklearn.model_selection import train_test_split
```

```
from scipy import stats
from scipy.stats.mstats import winsorize
from sklearn.experimental import enable_iterative_imputer # required to run
from sklearn.impute import IterativeImputer #Imputation of Nans
from sklearn.linear_model import BayesianRidge #Type of imputation
import smogn #Synthetic Minority Over-sampling Technique for Regression with
from sklearn.ensemble import RandomForestRegressor # Random forest model
from sklearn.feature_selection import SelectFromModel # to select the most i
from sklearn.model_selection import cross_val_score
from sklearn.metrics import mean_absolute_error, mean_squared_error, r2_scor
from sklearn import tree # Visualize the first tree
import optuna # Study for hyperparameter tuning
import joblib # Save outputs
import shap # SHapley Additive exPlanations
pd.options.mode.chained_assignment = None # Needed for SHAP
```

Import the data set using *pandas*

> **Note:** The output of the following cell is an expected *warning* on memory, *not an error*.

In [2]:
```
#Import data set as a pandas data frame
df_hypox = pd.read_csv('/Users/petrabier/PycharmProjects/Capstone_data/bold_
df_hypox.head()
```

```
/var/folders/v1/tqzcgz_96q16zh33lz92brt40000gq/T/ipykernel_17665/289641835
4.py:2: DtypeWarning: Columns (3) have mixed types. Specify dtype option on
import or set low_memory=False.
  df_hypox = pd.read_csv('/Users/petrabier/PycharmProjects/Capstone_data/bol
d_dataset.csv')
```

Out[2]:

	unique_subject_id	unique_hospital_admission_id	unique_icustay_id	subject_id	ho
0	0	0	0	002-10050	
1	1	1	1	002-1007	
2	2	2	2	002-10187	
3	3	3	3	002-10306	
4	4	4	4	002-10324	

5 rows × 142 columns

After the data was imported, it was explored to understand the variables.

In [3]:
```
df_hypox.describe()
```

Out[3]:

	unique_subject_id	unique_hospital_admission_id	unique_icustay_id	hospital_a
count	49093.000000	49093.000000	49093.000000	4.9
mean	22321.346770	24546.000000	24546.000000	3.
std	12867.699928	14172.072719	14172.072719	7.
min	0.000000	0.000000	0.000000	1.(
25%	11127.000000	12273.000000	12273.000000	7.
50%	22383.000000	24546.000000	24546.000000	1.
75%	33416.000000	36819.000000	36819.000000	2.4
max	44901.000000	49092.000000	49092.000000	2.9

8 rows × 129 columns

In [4]:
```
df_hypox.shape
```

Out[4]:
```
(49093, 142)
```

In [5]:
```
df_hypox.info
```

Out[5]:
```
<bound method DataFrame.info of       unique_subject_id  unique_hospital_a
dmission_id  unique_icustay_id  \
0                     0                           0                 0
1                     1                           1                 1
2                     2                           2                 2
3                     3                           3                 3
4                     4                           4                 4
...                 ...                         ...               ...
49088             44897                       49088             49088
49089             44898                       49089             49089
49090             44899                       49090             49090
49091             44900                       49091             49091
49092             44901                       49092             49092

      subject_id  hospital_admission_id  icustay_id source_db  hospitalid
\
0      002-10050                 183274      211144      eicu          71
1       002-1007                 178462      204935      eicu          71
2      002-10187                 150828      169525      eicu          73
3      002-10306                 198249      230427      eicu          63
4      002-10324                 188445      217835      eicu          73
...          ...                    ...         ...       ...         ...
49088   19995595               21784060    34670930  mimic_iv        9999
49089   19995780               21942461    36805359  mimic_iv        9999
49090   19997293               28847872    31877557  mimic_iv        9999
49091   19997367               20617667    35616526  mimic_iv        9999
49092   19997752               29452285    34531437  mimic_iv        9999

      numbedscategory  teachingstatus  ... delta_sofa_future_coagulation_24
hr  \
0           100 - 249           False  ...                              1525
.0
1           100 - 249           False  ...                                 N
aN
2              >= 500            True  ...                              1547
.0
3           100 - 249           False  ...                              1507
.0
4              >= 500            True  ...                              1537
.0
...               ...             ...  ...                                 .
..
49088          >= 500            True  ...                              1500
.0
49089          >= 500            True  ...                              1557
.0
49090          >= 500            True  ...                              1557
.0
49091          >= 500            True  ...                              1557
.0
49092          >= 500            True  ...                              1559
.0

       sofa_future_coagulation_24hr  delta_sofa_future_liver_24hr  \
0                               1.0                        1525.0
1                               NaN                           NaN
```

```
2                              0.0                          1547.0
3                              2.0                          1507.0
4                              1.0                          1537.0
...                            ...                             ...
49088                          0.0                          1500.0
49089                          2.0                          1557.0
49090                          0.0                          1557.0
49091                          3.0                          1557.0
49092                          0.0                          1559.0

       sofa_future_liver_24hr  delta_sofa_future_cardiovascular_24hr  \
0                         0.0                                1525.0
1                         NaN                                   NaN
2                         0.0                                1547.0
3                         0.0                                1507.0
4                         0.0                                1537.0
...                       ...                                   ...
49088                     2.0                                1500.0
49089                     0.0                                1557.0
49090                     0.0                                1557.0
49091                     2.0                                1557.0
49092                     0.0                                1559.0

       sofa_future_cardiovascular_24hr delta_sofa_future_cns_24hr  \
0                                  1.0                      1525.0
1                                  NaN                         NaN
2                                  1.0                      1547.0
3                                  1.0                      1507.0
4                                  1.0                      1537.0
...                                ...                         ...
49088                              1.0                      1500.0
49089                              1.0                      1557.0
49090                              1.0                      1557.0
49091                              1.0                      1557.0
49092                              1.0                      1559.0

      sofa_future_cns_24hr delta_sofa_future_renal_24hr sofa_future_renal_2
4hr
0                      0.0                       1525.0
0.0
1                      NaN                          NaN
NaN
2                      0.0                       1547.0
0.0
3                      0.0                       1507.0
1.0
4                      2.0                       1537.0
0.0
...                    ...                          ...
...
49088                  0.0                       1500.0
1.0
49089                  2.0                       1557.0
0.0
49090                  1.0                       1557.0
1.0
```

```
49091                  1.0                       1557.0
1.0
49092                  1.0                       1559.0
0.0

[49093 rows x 142 columns]>
```

In [6]:
```
df_hypox.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 49093 entries, 0 to 49092
Columns: 142 entries, unique_subject_id to sofa_future_renal_24hr
dtypes: bool(1), float64(122), int64(7), object(12)
memory usage: 52.9+ MB
```

In [7]:
```
list(df_hypox.columns.values)
```

Out[7]:
```
['unique_subject_id',
 'unique_hospital_admission_id',
 'unique_icustay_id',
 'subject_id',
 'hospital_admission_id',
 'icustay_id',
 'source_db',
 'hospitalid',
 'numbedscategory',
 'teachingstatus',
 'region',
 'admission_age',
 'sex_female',
 'weight_admission',
 'height_admission',
 'BMI_admission',
 'datetime_hospital_admit',
 'datetime_hospital_discharge',
 'datetime_icu_admit',
 'datetime_icu_discharge',
 'los_hospital',
 'los_ICU',
 'comorbidity_score_name',
 'comorbidity_score_value',
 'in_hospital_mortality',
 'race_ethnicity',
 'SaO2_timestamp',
 'pH',
 'pCO2',
 'pO2',
 'SaO2',
 'SpO2',
 'Carboxyhemoglobin',
 'Methemoglobin',
 'SpO2_timestamp',
 'delta_SpO2',
 'delta_vitals_heart_rate',
 'vitals_heart_rate',
 'delta_vitals_resp_rate',
 'vitals_resp_rate',
 'delta_vitals_mbp_ni',
 'vitals_mbp_ni',
 'delta_vitals_sbp_ni',
 'vitals_sbp_ni',
 'delta_vitals_dbp_ni',
 'vitals_dbp_ni',
 'delta_vitals_mbp_i',
 'vitals_mbp_i',
 'delta_vitals_sbp_i',
 'vitals_sbp_i',
 'delta_vitals_dbp_i',
 'vitals_dbp_i',
 'delta_vitals_tempc',
 'vitals_tempc',
 'delta_cbc_hemoglobin',
 'cbc_hemoglobin',
```

```
'delta_cbc_hematocrit',
'cbc_hematocrit',
'delta_cbc_mch',
'cbc_mch',
'delta_cbc_mchc',
'cbc_mchc',
'delta_cbc_mcv',
'cbc_mcv',
'delta_cbc_platelet',
'cbc_platelet',
'delta_cbc_rbc',
'cbc_rbc',
'delta_cbc_rdw',
'cbc_rdw',
'delta_cbc_wbc',
'cbc_wbc',
'delta_coag_fibrinogen',
'coag_fibrinogen',
'delta_coag_inr',
'coag_inr',
'delta_coag_pt',
'coag_pt',
'delta_coag_ptt',
'coag_ptt',
'delta_bmp_sodium',
'bmp_sodium',
'delta_bmp_potassium',
'bmp_potassium',
'delta_bmp_chloride',
'bmp_chloride',
'delta_bmp_bicarbonate',
'bmp_bicarbonate',
'delta_bmp_bun',
'bmp_bun',
'delta_bmp_creatinine',
'bmp_creatinine',
'delta_bmp_glucose',
'bmp_glucose',
'delta_bmp_aniongap',
'bmp_aniongap',
'delta_bmp_calcium',
'bmp_calcium',
'delta_bmp_lactate',
'bmp_lactate',
'delta_hfp_alt',
'hfp_alt',
'delta_hfp_alp',
'hfp_alp',
'delta_hfp_ast',
'hfp_ast',
'delta_hfp_bilirubin_total',
'hfp_bilirubin_total',
'delta_hfp_bilirubin_direct',
'hfp_bilirubin_direct',
'delta_hfp_albumin',
'hfp_albumin',
```

```
 'delta_others_ck_cpk',
 'others_ck_cpk',
 'delta_others_ck_mb',
 'others_ck_mb',
 'delta_others_ld_ldh',
 'others_ld_ldh',
 'delta_sofa_past_overall_24hr',
 'sofa_past_overall_24hr',
 'delta_sofa_past_coagulation_24hr',
 'sofa_past_coagulation_24hr',
 'delta_sofa_past_liver_24hr',
 'sofa_past_liver_24hr',
 'delta_sofa_past_cardiovascular_24hr',
 'sofa_past_cardiovascular_24hr',
 'delta_sofa_past_cns_24hr',
 'sofa_past_cns_24hr',
 'delta_sofa_past_renal_24hr',
 'sofa_past_renal_24hr',
 'delta_sofa_future_overall_24hr',
 'sofa_future_overall_24hr',
 'delta_sofa_future_coagulation_24hr',
 'sofa_future_coagulation_24hr',
 'delta_sofa_future_liver_24hr',
 'sofa_future_liver_24hr',
 'delta_sofa_future_cardiovascular_24hr',
 'sofa_future_cardiovascular_24hr',
 'delta_sofa_future_cns_24hr',
 'sofa_future_cns_24hr',
 'delta_sofa_future_renal_24hr',
 'sofa_future_renal_24hr']
```

In [8]:
```
print(df_hypox.duplicated().sum())
print(df_hypox.duplicated().value_counts())
```

```
0
False    49093
Name: count, dtype: int64
```

The sparsity of the data was examined.

In [9]:
```
df_hypox.isna().sum()
```

Out[9]:
```
unique_subject_id                    0
unique_hospital_admission_id         0
unique_icustay_id                    0
subject_id                           0
hospital_admission_id                0
                                   ...
sofa_future_cardiovascular_24hr    9426
delta_sofa_future_cns_24hr         9426
sofa_future_cns_24hr               9426
delta_sofa_future_renal_24hr       9426
sofa_future_renal_24hr             9426
Length: 142, dtype: int64
```

In [10]:
```
def analyze_nulls(df):
```

```
    total_nulls = df.isna().sum().sum()
    total_cells = df.size
    overall_null_percentage = round((total_nulls / total_cells) * 100, 2)
    print(f"\nTotal number of NaN values in the DataFrame: {total_nulls}")
    print(f"Overall percentage of NaN values: {overall_null_percentage}%")
    return total_nulls, overall_null_percentage
```

In [11]:
```
analyze_nulls(df_hypox)
```

```
Total number of NaN values in the DataFrame: 1719765
Overall percentage of NaN values: 24.67%
```

Out[11]: (1719765, 24.67)

All columns that will not be used in the analysis were dropped. Variables that are non-influential on the hypoxia gap calculation, such as mortality, were removed. The focus was on using laboratory values that are easily and commonly obtained with the intent of creating a model that is useful in rural hospitals and clinics.

No "invasive" measurements were included — no arterial blood gas values or arterial vital signs were used. SaO2 was kept for the hidden hypoxia gap measurement and will not be used as a predictor in the RFR model.

In [12]:
```
df_clean = df_hypox[['admission_age','sex_female','weight_admission', 'heigh
```

In [13]:
```
# Verify that columns have been dropped
df_clean.head()
```

Out[13]:

	admission_age	sex_female	weight_admission	height_admission	BMI_admission	r
0	67.0	1	86.2	160.0	33.671875	
1	83.0	1	NaN	162.6	NaN	
2	59.0	1	74.1	162.6	28.027033	
3	73.0	0	NaN	152.4	NaN	
4	57.0	0	NaN	172.7	NaN	

5 rows × 65 columns

Visualize the missing data

Missing data was visualized using the *matplotlib.pyplot* and the *seaborn* library.

Return to C1: Data Extraction and Preparation , Table of Contents

In [14]:
```
def plot_missing_data(df):
    plt.figure(figsize=(12, 8))
    sns.heatmap(
        df.isnull(),
        cmap='BuGn',
```

```
        yticklabels=False,
        cbar=False
    )
    plt.title('Missing Data Visualization')
    plt.xticks(rotation=45, ha='right')
    plt.tight_layout()
    plt.show()

plot_missing_data(df_clean)
```

In [15]:
```
df_clean.describe()
```

Out[15]:

	admission_age	sex_female	weight_admission	height_admission	BMI_admis
count	49091.000000	49093.000000	47989.000000	47809.000000	47108.00
mean	64.429549	0.436580	85.611319	169.347372	200.28
std	15.800227	0.495967	27.847338	12.872780	8189.04
min	14.000000	0.000000	0.000000	0.000000	0.00
25%	55.000000	0.000000	67.400000	162.560000	23.82
50%	66.000000	0.000000	81.500000	170.000000	28.05
75%	76.000000	1.000000	99.000000	177.800000	33.56
max	90.000000	1.000000	771.200000	504.800000	725551.02

8 rows × 64 columns

In [16]:
```
list(df_clean.columns.values)
```

Out[16]:
```
['admission_age',
 'sex_female',
 'weight_admission',
 'height_admission',
 'BMI_admission',
 'race_ethnicity',
 'SaO2',
 'SpO2',
 'delta_vitals_heart_rate',
 'vitals_heart_rate',
 'delta_vitals_resp_rate',
 'vitals_resp_rate',
 'delta_vitals_mbp_ni',
 'vitals_mbp_ni',
 'delta_vitals_sbp_ni',
 'vitals_sbp_ni',
 'delta_vitals_dbp_ni',
 'delta_vitals_tempc',
 'vitals_tempc',
 'delta_cbc_hemoglobin',
 'cbc_hemoglobin',
 'delta_cbc_hematocrit',
 'cbc_hematocrit',
 'delta_cbc_mch',
 'cbc_mch',
 'delta_cbc_mchc',
 'cbc_mchc',
 'delta_cbc_mcv',
 'cbc_mcv',
 'delta_cbc_platelet',
 'cbc_platelet',
 'delta_cbc_rbc',
 'cbc_rbc',
 'delta_cbc_rdw',
 'cbc_rdw',
 'delta_cbc_wbc',
 'cbc_wbc',
 'delta_bmp_sodium',
 'bmp_sodium',
 'delta_bmp_potassium',
 'bmp_potassium',
 'delta_bmp_chloride',
 'bmp_chloride',
 'delta_bmp_bicarbonate',
 'bmp_bicarbonate',
 'delta_bmp_bun',
 'bmp_bun',
 'delta_bmp_creatinine',
 'bmp_creatinine',
 'delta_bmp_glucose',
 'bmp_glucose',
 'delta_bmp_aniongap',
 'bmp_aniongap',
 'delta_bmp_calcium',
 'bmp_calcium',
 'delta_bmp_lactate',
```

```
 'bmp_lactate',
 'delta_hfp_alt',
 'hfp_alt',
 'delta_hfp_alp',
 'hfp_alp',
 'delta_hfp_ast',
 'hfp_ast',
 'delta_hfp_albumin',
 'hfp_albumin']
```

In [17]:

```
# Check dataframe info
print("DataFrame Info:")
df_clean.info()
# Check if it is a copy or view
print("\nIs df_clean a view?")
print(df_clean._is_view)
```

```
DataFrame Info:
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 49093 entries, 0 to 49092
Data columns (total 65 columns):
 #   Column                   Non-Null Count  Dtype
---  ------                   --------------  -----
 0   admission_age            49091 non-null  float64
 1   sex_female               49093 non-null  int64
 2   weight_admission         47989 non-null  float64
 3   height_admission         47809 non-null  float64
 4   BMI_admission            47108 non-null  float64
 5   race_ethnicity           49093 non-null  object
 6   SaO2                     49093 non-null  float64
 7   SpO2                     49093 non-null  float64
 8   delta_vitals_heart_rate  43643 non-null  float64
 9   vitals_heart_rate        43643 non-null  float64
 10  delta_vitals_resp_rate   42217 non-null  float64
 11  vitals_resp_rate         42217 non-null  float64
 12  delta_vitals_mbp_ni      33626 non-null  float64
 13  vitals_mbp_ni            33626 non-null  float64
 14  delta_vitals_sbp_ni      34141 non-null  float64
 15  vitals_sbp_ni            34141 non-null  float64
 16  delta_vitals_dbp_ni      34137 non-null  float64
 17  delta_vitals_tempc       41134 non-null  float64
 18  vitals_tempc             41134 non-null  float64
 19  delta_cbc_hemoglobin     42297 non-null  float64
 20  cbc_hemoglobin           42297 non-null  float64
 21  delta_cbc_hematocrit     42360 non-null  float64
 22  cbc_hematocrit           42360 non-null  float64
 23  delta_cbc_mch            38072 non-null  float64
 24  cbc_mch                  38072 non-null  float64
 25  delta_cbc_mchc           39843 non-null  float64
 26  cbc_mchc                 39843 non-null  float64
 27  delta_cbc_mcv            39850 non-null  float64
 28  cbc_mcv                  39850 non-null  float64
 29  delta_cbc_platelet       41707 non-null  float64
 30  cbc_platelet             41707 non-null  float64
 31  delta_cbc_rbc            41232 non-null  float64
 32  cbc_rbc                  41232 non-null  float64
 33  delta_cbc_rdw            38155 non-null  float64
 34  cbc_rdw                  38155 non-null  float64
 35  delta_cbc_wbc            41357 non-null  float64
 36  cbc_wbc                  41357 non-null  float64
 37  delta_bmp_sodium         42554 non-null  float64
 38  bmp_sodium               42554 non-null  float64
 39  delta_bmp_potassium      43128 non-null  float64
 40  bmp_potassium            43128 non-null  float64
 41  delta_bmp_chloride       41186 non-null  float64
 42  bmp_chloride             41186 non-null  float64
 43  delta_bmp_bicarbonate    38426 non-null  float64
 44  bmp_bicarbonate          38426 non-null  float64
 45  delta_bmp_bun            41338 non-null  float64
 46  bmp_bun                  41338 non-null  float64
 47  delta_bmp_creatinine     41297 non-null  float64
 48  bmp_creatinine           41297 non-null  float64
 49  delta_bmp_glucose        44359 non-null  float64
```

```
 50  bmp_glucose              44359 non-null  float64
 51  delta_bmp_aniongap       35334 non-null  float64
 52  bmp_aniongap             35334 non-null  float64
 53  delta_bmp_calcium        38226 non-null  float64
 54  bmp_calcium              38226 non-null  float64
 55  delta_bmp_lactate        23120 non-null  float64
 56  bmp_lactate              23120 non-null  float64
 57  delta_hfp_alt            31498 non-null  float64
 58  hfp_alt                  31498 non-null  float64
 59  delta_hfp_alp            31237 non-null  float64
 60  hfp_alp                  31237 non-null  float64
 61  delta_hfp_ast            31634 non-null  float64
 62  hfp_ast                  31634 non-null  float64
 63  delta_hfp_albumin        31423 non-null  float64
 64  hfp_albumin              31423 non-null  float64
dtypes: float64(63), int64(1), object(1)
memory usage: 24.3+ MB

Is df_clean a view?
False
```

Categorical variables were explored using both *NumPy* and *Pandas* libraries.

In [18]:
```
print(df_clean['sex_female'].unique())
print(df_clean['sex_female'].value_counts())
```

```
[1 0]
sex_female
0    27660
1    21433
Name: count, dtype: int64
```

In [19]:
```
print(df_clean['race_ethnicity'].unique())
print(df_clean['race_ethnicity'].value_counts())
```

```
['White' 'Asian' 'Black' 'Unknown' 'American Indian / Alaska Native'
 'Hispanic OR Latino' 'Native Hawaiian / Pacific Islander'
 'More Than One Race']
race_ethnicity
White                                  37380
Black                                   4785
Unknown                                 3562
Hispanic OR Latino                      2116
Asian                                    858
American Indian / Alaska Native          380
Native Hawaiian / Pacific Islander         9
More Than One Race                         3
Name: count, dtype: int64
```

All columns with a *delta* label were dropped. These columns measured the difference in time that variable measurements were taken in relation to the original SaO2 time.

In [20]:
```
delta_columns = [col for col in df_clean.columns if 'delta' in col.lower()]
print("Delta columns found:", delta_columns)
```

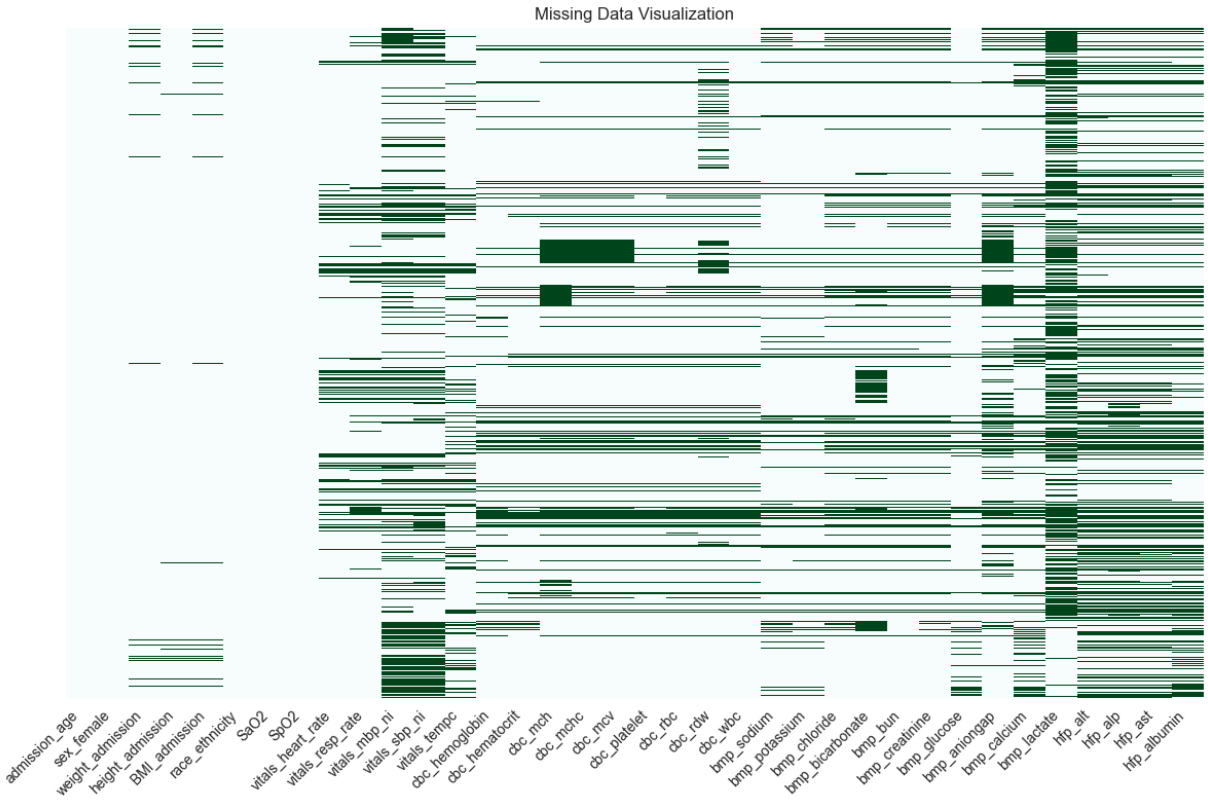
```
Delta columns found: ['delta_vitals_heart_rate', 'delta_vitals_resp_rate', '
delta_vitals_mbp_ni', 'delta_vitals_sbp_ni', 'delta_vitals_dbp_ni', 'delta_v
itals_tempc', 'delta_cbc_hemoglobin', 'delta_cbc_hematocrit', 'delta_cbc_mch
', 'delta_cbc_mchc', 'delta_cbc_mcv', 'delta_cbc_platelet', 'delta_cbc_rbc',
'delta_cbc_rdw', 'delta_cbc_wbc', 'delta_bmp_sodium', 'delta_bmp_potassium',
'delta_bmp_chloride', 'delta_bmp_bicarbonate', 'delta_bmp_bun', 'delta_bmp_c
reatinine', 'delta_bmp_glucose', 'delta_bmp_aniongap', 'delta_bmp_calcium',
'delta_bmp_lactate', 'delta_hfp_alt', 'delta_hfp_alp', 'delta_hfp_ast', 'del
ta_hfp_albumin']
```

In [21]:

```
df_dropped = df_clean.drop(delta_columns, axis=1)
df_dropped.shape
df_dropped.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 49093 entries, 0 to 49092
Data columns (total 36 columns):
 #   Column             Non-Null Count  Dtype  
---  ------             --------------  -----  
 0   admission_age      49091 non-null  float64
 1   sex_female         49093 non-null  int64  
 2   weight_admission   47989 non-null  float64
 3   height_admission   47809 non-null  float64
 4   BMI_admission      47108 non-null  float64
 5   race_ethnicity     49093 non-null  object 
 6   SaO2               49093 non-null  float64
 7   SpO2               49093 non-null  float64
 8   vitals_heart_rate  43643 non-null  float64
 9   vitals_resp_rate   42217 non-null  float64
 10  vitals_mbp_ni      33626 non-null  float64
 11  vitals_sbp_ni      34141 non-null  float64
 12  vitals_tempc       41134 non-null  float64
 13  cbc_hemoglobin     42297 non-null  float64
 14  cbc_hematocrit     42360 non-null  float64
 15  cbc_mch            38072 non-null  float64
 16  cbc_mchc           39843 non-null  float64
 17  cbc_mcv            39850 non-null  float64
 18  cbc_platelet       41707 non-null  float64
 19  cbc_rbc            41232 non-null  float64
 20  cbc_rdw            38155 non-null  float64
 21  cbc_wbc            41357 non-null  float64
 22  bmp_sodium         42554 non-null  float64
 23  bmp_potassium      43128 non-null  float64
 24  bmp_chloride       41186 non-null  float64
 25  bmp_bicarbonate    38426 non-null  float64
 26  bmp_bun            41338 non-null  float64
 27  bmp_creatinine     41297 non-null  float64
 28  bmp_glucose        44359 non-null  float64
 29  bmp_aniongap       35334 non-null  float64
 30  bmp_calcium        38226 non-null  float64
 31  bmp_lactate        23120 non-null  float64
 32  hfp_alt            31498 non-null  float64
 33  hfp_alp            31237 non-null  float64
 34  hfp_ast            31634 non-null  float64
 35  hfp_albumin        31423 non-null  float64
dtypes: float64(34), int64(1), object(1)
memory usage: 13.5+ MB
```

In [22]:
```
plot_missing_data(df_dropped)
```



In [23]:
```
# Examine smaller dataframe
df_dropped.head()
```

Out[23]:

	admission_age	sex_female	weight_admission	height_admission	BMI_admission	r
0	67.0	1	86.2	160.0	33.671875	
1	83.0	1	NaN	162.6	NaN	
2	59.0	1	74.1	162.6	28.027033	
3	73.0	0	NaN	152.4	NaN	
4	57.0	0	NaN	172.7	NaN	

5 rows × 36 columns

In [24]:
```
df_dropped.shape
```

Out[24]: (49093, 36)

C2: Data Preprocessing

The data required preprocessing before it could be used in the RFR model. The majority of preprocessing was done with the SciKit – Learn package. This package is well established and works well with Pandas and NumPy, although it is known to prioritize ease of use over performance.

Further examination was done through visualizing the distribution using plot

distributions and quantile-quantile(Q-Q) plots. Q-Q plots show if the data has a normal distribution. Large data sets may not always have normal data, since the large number of values can actually skew the data (Varshney, 2023). Random Forest models work well with data that does not have a normal distribution and can look for non-linear relationships between data points. The data was split early in the analysis to prevent data leakage (Brownlee, 2020). This function allowed for the splitting of the data using a single line of code. A disadvantage of using this function is that it may not be well suited to imbalanced data sets.

- from sklearn.model_selection import train_test_split

Outliers were identified and managed using winsorization. This was done using the SciPy package. The winsorization technique maintains the statistical power of the data while dealing with extreme outliers. To set the thresholds at appropriate values, it may be necessary to evaluate the variables individually to reduce the risk of imposing arbitrary values. This is a disadvantage to this technique as it can be time-consuming and requires a high level of domain knowledge.

- from scipy import stats
- from scipy.stats.mstats import winsorize

The data was found to be imbalancedBalance) meaning there is a significantly unequal distribution between the target variable and the features. This presents a problem as the model will be biased toward the majority since it has little data to learn from about the minority cases. One method used to address this problem was stratifying the data during the train-test split. This helps to keep the proportion of cases as they are originally distributed (Roepke, 2024). This ensures that when the data is split the minority class is not left out of the training data, which is a possibility when splitting the data randomly. A problem that can occur with this technique is that the data is not a true representation of real-world data, which can reduce the strength of the model. For stratification purposes, the data was classified based on gap values of three or greater. This threshold was chosen because it will help identify patients with true hypoxemia ($SaO_2 \leq 90\%$) even when their pulse oximetry readings appear normal ($SpO_2 \geq 93\%$).

It should be noted that the objective here is to find gap values greater than three, which can easily be confused with the RMSE metric where values of three or less are necessary to reject the null hypothesis.

Once the data was stratified and split, the missing values were filled using imputation. The SciKit-Learn package has an iterative imputer function. Imputation is a technique that looks at relationships between values to estimate the missing values rather than simply using mean or median values. The advantage to this is that the new values can help preserve the overall statistical properties within the data. A downside to this method is that it can be time-consuming with larger data sets. The Bayesian method was chosen for imputation since it can handle multicollinearity. A drawback of this

method is that it is also computationally expensive and may not capture non-linear relationships.

- from sklearn.experimental import enable_iterative_imputer
- from sklearn.impute import IterativeImputer
- from sklearn.linear_model import BayesianRidge

The categorical data was encoded into numerical values for the analysis. The columns with low cardinality were one-hot encoded with all columns kept. If the column had more than five unique variables, the variables were target encoded. This is a technique that replaces categorical variables with the mean of the target variable for each category. The main advantage to this technique is that it effectively manages high cardinality variables. A downside is that there is a risk of overfitting if there are rare categories.

Over-sampling and under-sampling methods can also be used in an attempt to balance out the data, so that the trained model has more exposure to the minority outcomes, in turn leading to better predictions. For this analysis, two separate models were built, one of which used the Synthetic Minority Over-sampling Technique for Regression with Gaussian Noise (SMOGN) method, and one that did not. The SMOGN package uses an oversampling technique Synthetic Minority Over-sampling Technique for Regression (SMOTER) that creates synthetic data to help balance out the data for more robust training of regression models (Kunz, 2020). A large benefit to SMOTER is that it combines over-sampling of the minority class with under-sampling of majority instances to create a balance between classes in an attempt to improve predictions. The main drawback of SMOTER is the computational time that was needed for a data set this large. Due to the time and computational constraints of using this technique (approximately six hours of run time), the final RFR model does not use SMOTER in the analysis. The model which did have this technique can be viewed in the Appendix.

Distribution plots

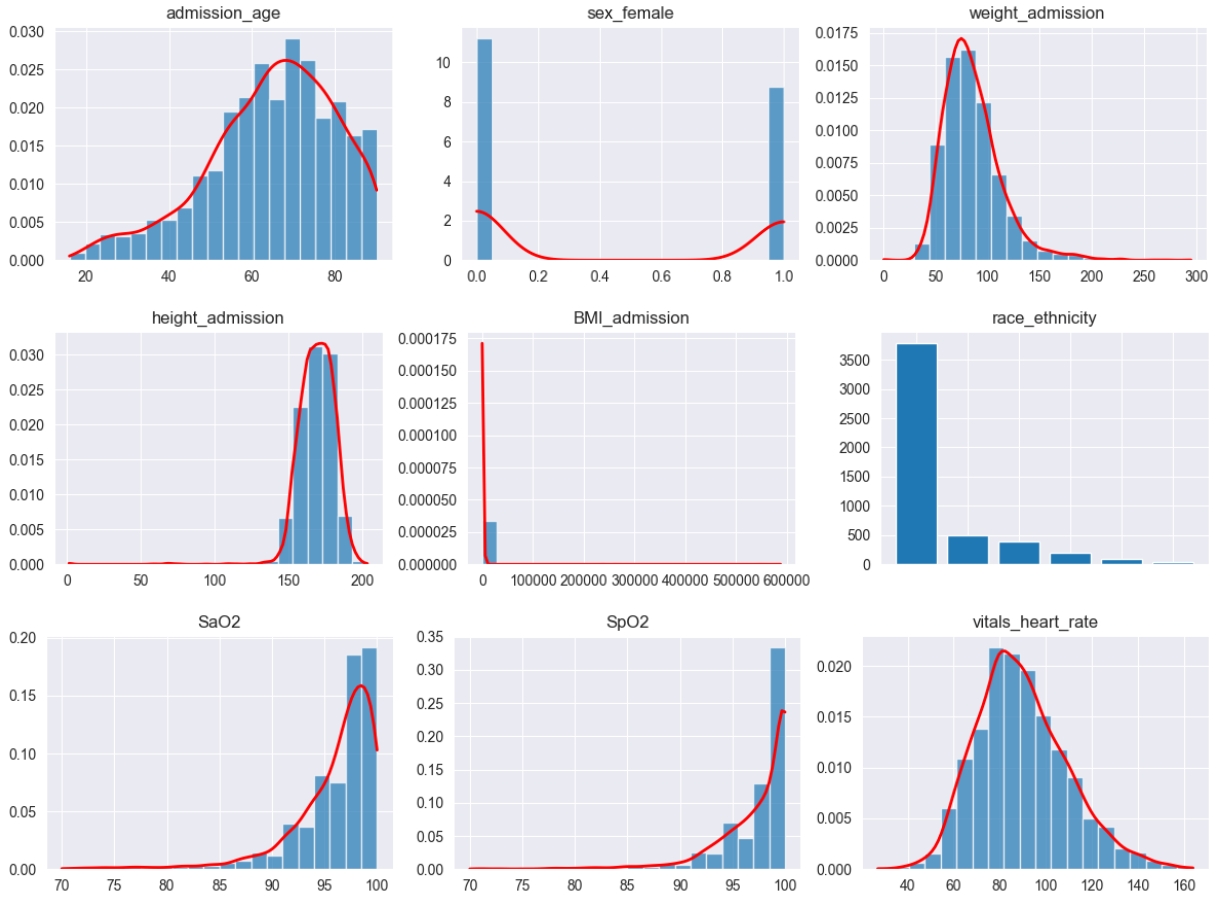
Distribution of the final columns was visualized.

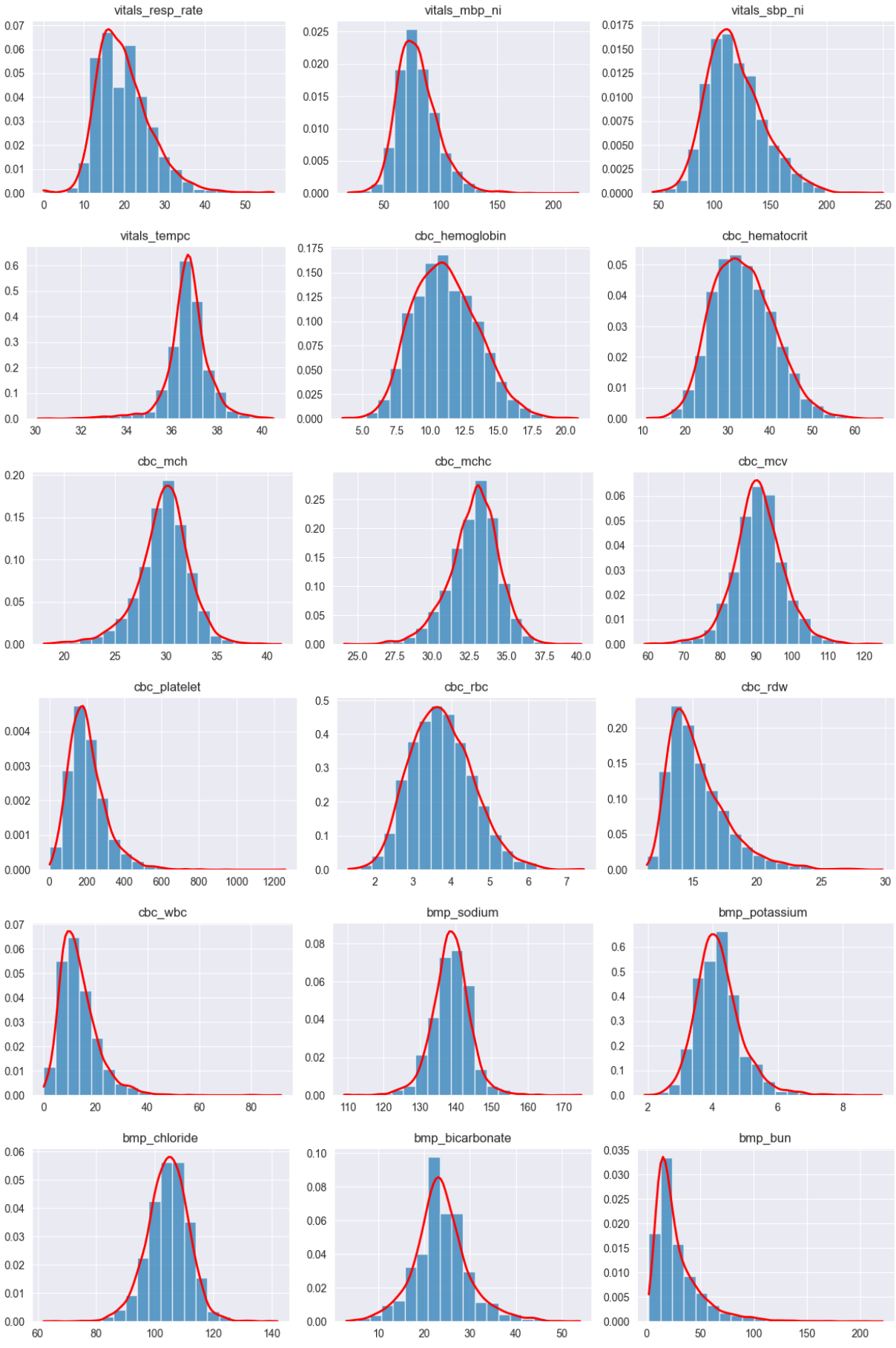
Return to C2: Data Preprocessing , Table of Contents

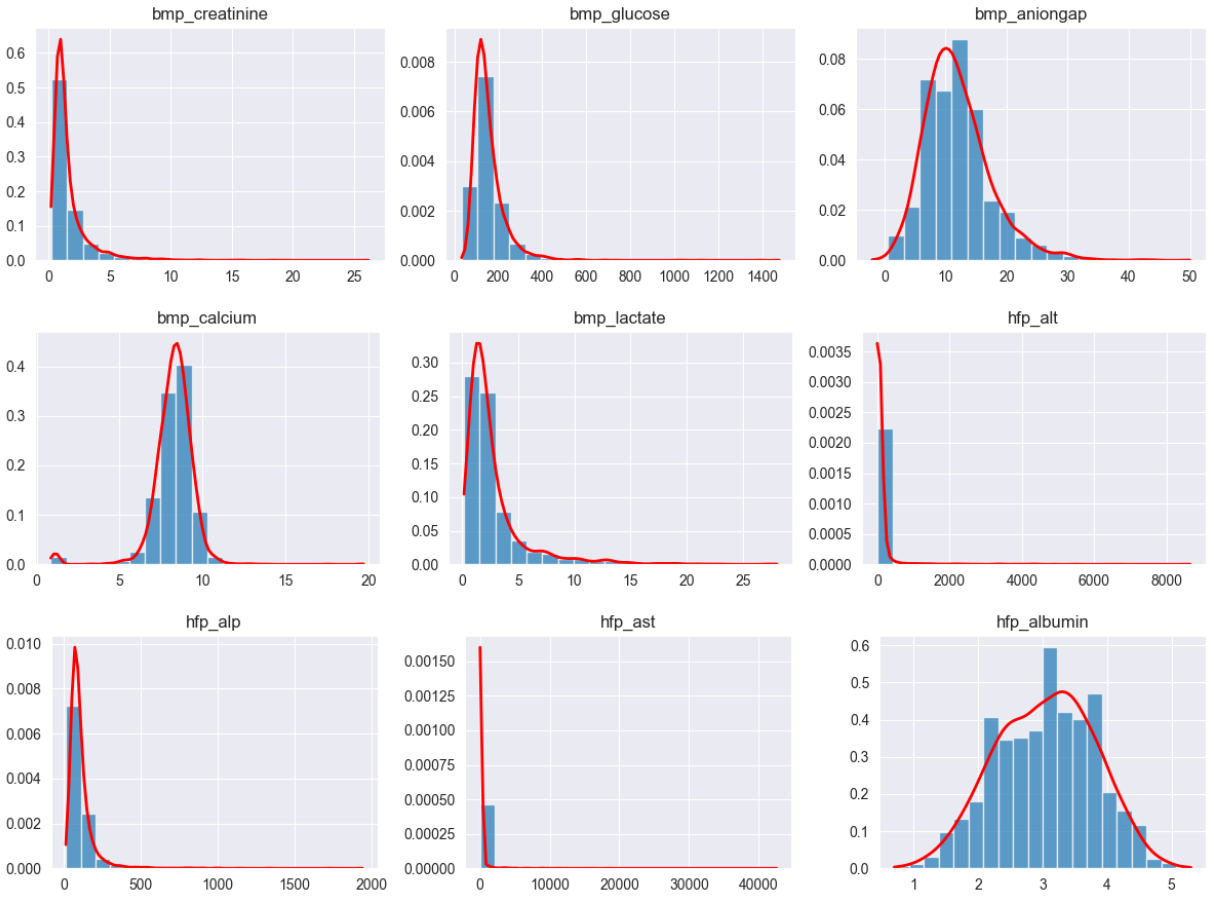
In [25]:
```
#Create function to plot the distributions of columns
def plot_distributions(df, n_cols=3, figsize=(12, 3), max_rows=5000):
    plot_df = df.sample(min(max_rows, len(df))) if len(df) > max_rows else d
    numeric_cols = list(plot_df.select_dtypes(include=['int64', 'float64']).
    # Process in batches
    for i in range(0, len(df.columns), n_cols):
        batch_cols = list(df.columns[i:i+n_cols])
        n_plots = len(batch_cols)

        fig, axes = plt.subplots(1, n_plots, figsize=figsize)
        if n_plots == 1:
            axes = [axes]
```

```
        for j, col in enumerate(batch_cols):
            ax = axes[j]

            if col in numeric_cols:
                clean_data = plot_df[col].dropna().values

                if len(clean_data) > 0:
                    ax.hist(clean_data, bins=20, density=True, alpha=0.7)
                    if len(clean_data) > 1:
                        x = np.linspace(min(clean_data), max(clean_data), 10
                        kde = stats.gaussian_kde(clean_data)
                        ax.plot(x, kde(x), 'r-', linewidth=2)
            else:
                counts = plot_df[col].value_counts().iloc[:10]
                ax.bar(range(len(counts)), counts.values)
                ax.set_xticks(range(len(counts)))
                ax.set_xticklabels([])  # Skip labels entirely for speed

            ax.set_title(col)

        plt.tight_layout()
        plt.show()
        plt.close(fig)  # Important for memory
```

In [26]:
```
#Visualize distributions
plot_distributions(df_dropped)
```




vitals_resp_rate
vitals_mbp_ni
vitals_sbp_ni
vitals_tempc
cbc_hemoglobin
cbc_hematocrit
cbc_mch
cbc_mchc
cbc_mcv
cbc_platelet
cbc_rbc
cbc_rdw
cbc_wbc
bmp_sodium
bmp_potassium
bmp_chloride
bmp_bicarbonate
bmp_bun



In [27]:
```
# Count unique values in each column using nunique()
column_count = df_dropped.nunique()
pd.set_option('display.max_columns', None)
pd.set_option('display.max_rows', None)
pd.set_option('display.width', None)
print("Number of unique values in each column:\n", column_count)
```

```
Number of unique values in each column:
 admission_age           77
sex_female               2
weight_admission      2833
height_admission       512
BMI_admission        25760
race_ethnicity           8
SaO2                   291
SpO2                    31
vitals_heart_rate      163
vitals_resp_rate        91
vitals_mbp_ni          189
vitals_sbp_ni          205
vitals_tempc           495
cbc_hemoglobin         192
cbc_hematocrit         513
cbc_mch                257
cbc_mchc               139
cbc_mcv                585
cbc_platelet           767
cbc_rbc                567
cbc_rdw                208
cbc_wbc               2823
bmp_sodium             198
bmp_potassium          192
bmp_chloride            87
bmp_bicarbonate        308
bmp_bun                297
bmp_creatinine        1149
bmp_glucose            672
bmp_aniongap           317
bmp_calcium            180
bmp_lactate            767
hfp_alt               1257
hfp_alp                683
hfp_ast               1530
hfp_albumin             56
dtype: int64
```

In [28]:
```
df_dropped.describe()
```

Out[28]:

	admission_age	sex_female	weight_admission	height_admission	BMI_admis
count	49091.000000	49093.000000	47989.000000	47809.000000	47108.00(
mean	64.429549	0.436580	85.611319	169.347372	200.28
std	15.800227	0.495967	27.847338	12.872780	8189.04
min	14.000000	0.000000	0.000000	0.000000	0.00(
25%	55.000000	0.000000	67.400000	162.560000	23.82
50%	66.000000	0.000000	81.500000	170.000000	28.05
75%	76.000000	1.000000	99.000000	177.800000	33.56
max	90.000000	1.000000	771.200000	504.800000	725551.02(

The percentage of each column's sparsity was calculated to determine which columns will be kept.

In [29]:
```
# Get count of NaN values in each column
null_counts = df_dropped.isna().sum()
# Calculate the percentage of NaN values in each column
null_percentages = round((df_dropped.isna().sum() / len(df_dropped)) * 100,
# Combine counts and percentages in a DataFrame for better visualization
null_info = pd.DataFrame({
    'Null Count': null_counts,
    'Null Percentage': null_percentages
})

# Show only columns that have at least one NaN value
null_info_with_nulls = null_info[null_counts > 0]
# Print the results
print("Columns with missing values:")
print(null_info_with_nulls)
```

```
Columns with missing values:
                   Null Count  Null Percentage
admission_age               2             0.00
weight_admission         1104             2.25
height_admission         1284             2.62
BMI_admission            1985             4.04
vitals_heart_rate        5450            11.10
vitals_resp_rate         6876            14.01
vitals_mbp_ni           15467            31.51
vitals_sbp_ni           14952            30.46
vitals_tempc             7959            16.21
cbc_hemoglobin           6796            13.84
cbc_hematocrit           6733            13.71
cbc_mch                 11021            22.45
cbc_mchc                 9250            18.84
cbc_mcv                  9243            18.83
cbc_platelet             7386            15.04
cbc_rbc                  7861            16.01
cbc_rdw                 10938            22.28
cbc_wbc                  7736            15.76
bmp_sodium               6539            13.32
bmp_potassium            5965            12.15
bmp_chloride             7907            16.11
bmp_bicarbonate         10667            21.73
bmp_bun                  7755            15.80
bmp_creatinine           7796            15.88
bmp_glucose              4734             9.64
bmp_aniongap            13759            28.03
bmp_calcium             10867            22.14
bmp_lactate             25973            52.91
hfp_alt                 17595            35.84
hfp_alp                 17856            36.37
hfp_ast                 17459            35.56
hfp_albumin             17670            35.99
```

Columns with greater than 50% of missing values were excluded from the analysis. The lactate variable was removed from the data frame.

In [30]:
```
df_dropped = df_dropped.drop('bmp_lactate', axis=1)
```

In [31]:
```
# Verify removal
df_dropped.shape
```

Out[31]: (49093, 35)

In [32]:
```
# Check sparsity of data frame after lactate removal
analyze_nulls(df_dropped)
```

```
Total number of NaN values in the DataFrame: 278612
Overall percentage of NaN values: 16.21%
```

Out[32]: (278612, 16.21)

SaO2 and SpO2 variables were examined to determine if any outliers were present. This was done before creating the target variable column to ensure accurate data for the calculation. The original data set had specified the range of values to be between 70% -100%.

In [33]:
```
#Explore SaO2 values - no value should be above 100 or less than 70
df_dropped['SaO2'] = df_dropped['SaO2'].apply(lambda x: 100 if x >100 else x
df_dropped['SaO2'] = df_dropped['SaO2'].apply(lambda x: 70 if x < 70 else x)
df_dropped["SaO2"].describe()
```

Out[33]:
```
count    49093.000000
mean        95.917711
std          4.180759
min         70.000000
25%         94.600000
50%         97.000000
75%         99.000000
max        100.000000
Name: SaO2, dtype: float64
```

In [34]:
```
#Explore SpO2 values - no value should be above 100 or less than 70
df_dropped['SpO2'] = df_dropped['SpO2'].apply(lambda x: 100 if x >100 else x
df_dropped['SpO2'] = df_dropped['SpO2'].apply(lambda x: 70 if x < 70 else x)
df_dropped["SpO2"].describe()
```

Out[34]:
```
count    49093.000000
mean        97.137066
std          3.834850
min         70.000000
25%         96.000000
50%         98.000000
75%        100.000000
max        100.000000
Name: SpO2, dtype: float64
```

The target variable of gap was calculated and added to the data frame. Gap is the difference between the SpO2 and the SaO2 measurements.

In [35]:
```
# Add the measurement of the gap between SpO2 and SaO2
df_dropped.insert(5, 'gap', df_dropped['SpO2'] - df_dropped['SaO2'])
df_dropped.head(10)
```

Out[35]:

	admission_age	sex_female	weight_admission	height_admission	BMI_admission
0	67.0	1	86.2	160.0	33.671875
1	83.0	1	NaN	162.6	NaN
2	59.0	1	74.1	162.6	28.027033
3	73.0	0	NaN	152.4	NaN
4	57.0	0	NaN	172.7	NaN
5	59.0	0	116.7	175.3	37.975808
6	63.0	1	60.2	149.9	26.791265
7	63.0	1	84.8	154.9	35.342154
8	60.0	0	132.4	180.3	40.728323
9	60.0	0	125.9	172.7	42.212498

In [36]:

```
# More cleaning of the dataframe
# Make all lower case
df_dropped.columns = df_dropped.columns.str.lower()

for col in df_dropped.columns:
    if df_dropped[col].dtype == 'object':
        df_dropped[col] = df_dropped[col].str.lower()

#round values to 2 decimal points
df_lower = df_dropped.round(2)
#Verify space removal/ replacement
df_lower.head(10)
```

Out[36]:

	admission_age	sex_female	weight_admission	height_admission	bmi_admission
0	67.0	1	86.2	160.0	33.67
1	83.0	1	NaN	162.6	NaN
2	59.0	1	74.1	162.6	28.03
3	73.0	0	NaN	152.4	NaN
4	57.0	0	NaN	172.7	NaN
5	59.0	0	116.7	175.3	37.98
6	63.0	1	60.2	149.9	26.79
7	63.0	1	84.8	154.9	35.34
8	60.0	0	132.4	180.3	40.73
9	60.0	0	125.9	172.7	42.21

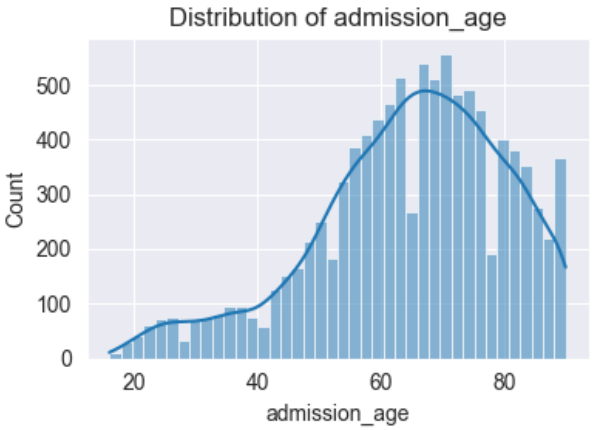
Q-Q Plots

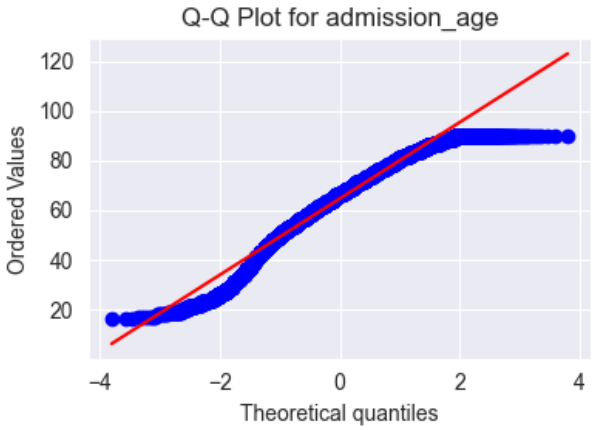
The normality of the data was visualized with Q-Q plots. Random forest regression models do not work on assumptions of normality and are able to handle data that is not normal. Often times in larger data sets, the data is found to be non-normally distributed (Varshney, 2023).

Return to C2: Data Preprocessing , Table of Contents

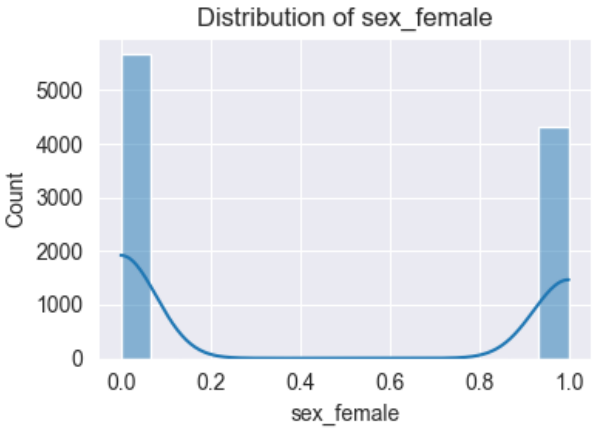
In [37]:
```
def check_normality(data, max_features=10):
    numeric_cols = data.select_dtypes('number').columns
    if len(numeric_cols) > max_features:
        print(f"Showing first {max_features} features out of {len(numeric_co
        numeric_cols = numeric_cols[:max_features]

    for col in numeric_cols:
        try:
            col_data = data[col].dropna()
            if len(col_data) < 10:
                print(f"{col}: Too few non-missing values")
                continue

            if len(col_data) > 10000:
                print(f"{col}: Sampling 10,000 points from {len(col_data)} t
                col_data = col_data.sample(10000, random_state=42)

            fig, (ax1, ax2) = plt.subplots(1, 2, figsize=(8, 3))
            sns.histplot(col_data, kde=True, ax=ax1)
            ax1.set_title(f'Distribution of {col}')
            stats.probplot(col_data, plot=ax2)
            ax2.set_title(f'Q-Q Plot for {col}')
            plt.tight_layout()
            plt.show()
            plt.close(fig)
            skew = col_data.skew()
            print(f"{col}: Skewness = {skew:.4f}")
            if abs(skew) < 0.5:
                print("  Approximately normal")
            else:
                print("  Not normal")

            print()

        except Exception as e:
            print(f"Error processing {col}: {str(e)}")
            continue

print(f"DataFrame shape: {df_lower.shape}")
check_normality(df_lower)
```

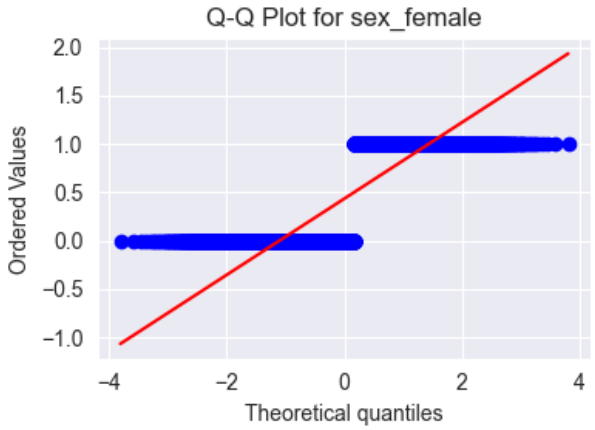
```
DataFrame shape: (49093, 36)
Showing first 10 features out of 35
admission_age: Sampling 10,000 points from 49091 total
```



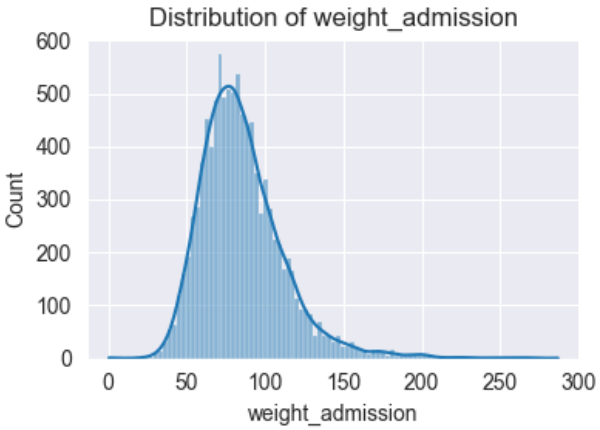


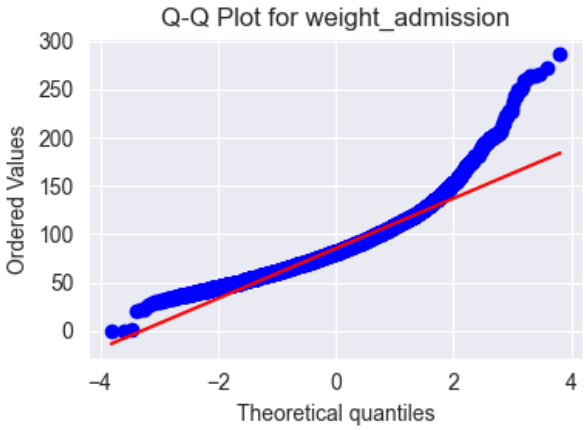

```
admission_age: Skewness = -0.6583
  Not normal

sex_female: Sampling 10,000 points from 49093 total
```



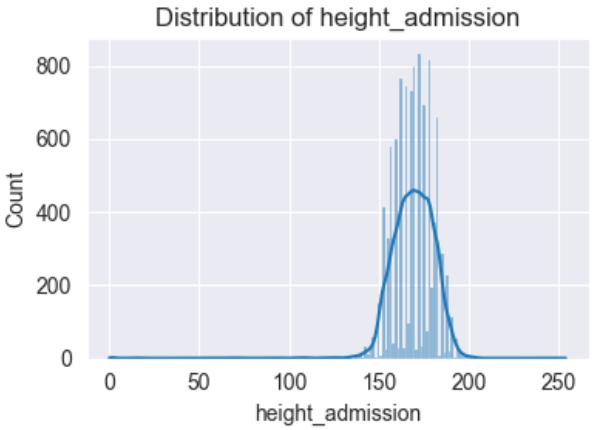


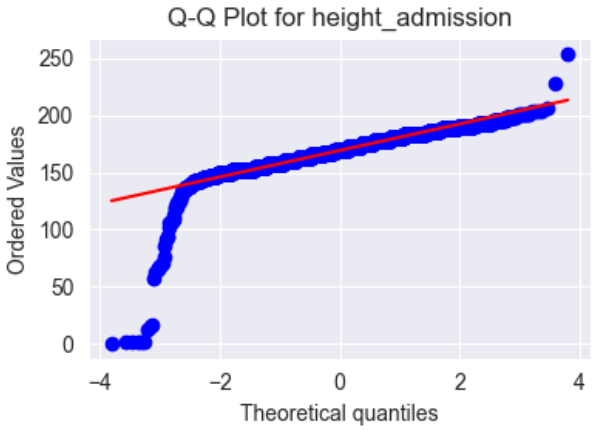

```
sex_female: Skewness = 0.2709
  Approximately normal

weight_admission: Sampling 10,000 points from 47989 total
```



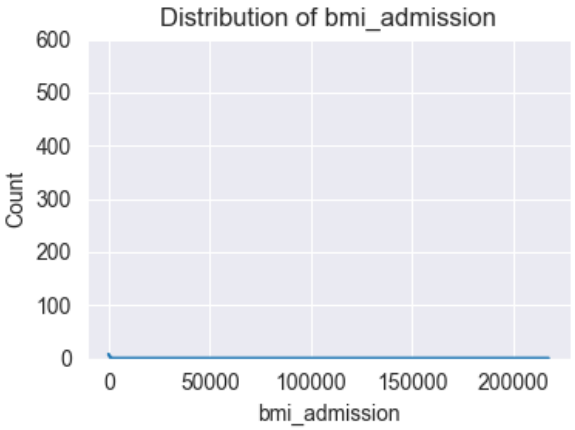


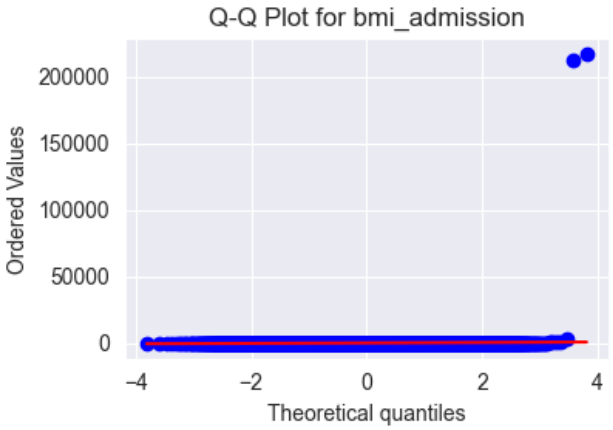

```
weight_admission: Skewness = 1.3863
  Not normal

height_admission: Sampling 10,000 points from 47809 total
```



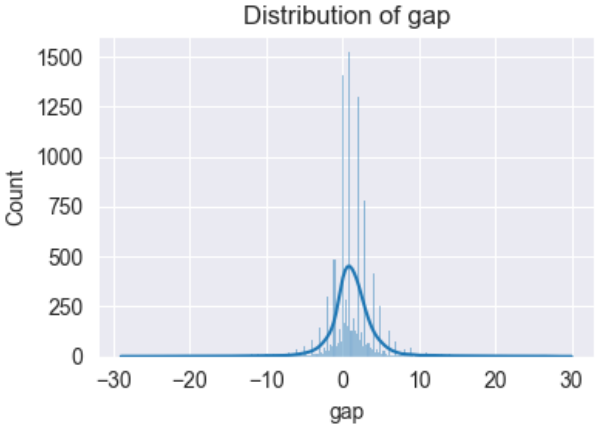


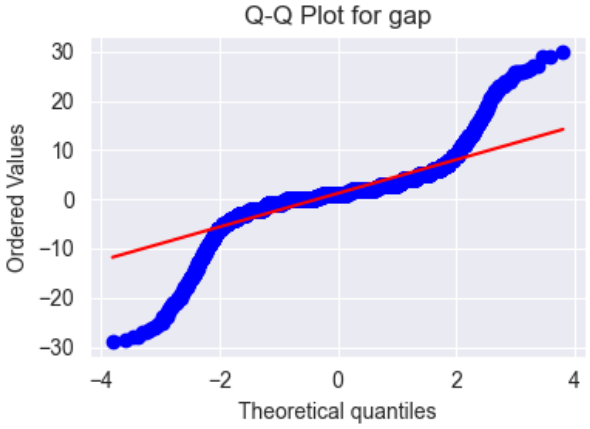

```
height_admission: Skewness = -2.6976
  Not normal

bmi_admission: Sampling 10,000 points from 47108 total
```



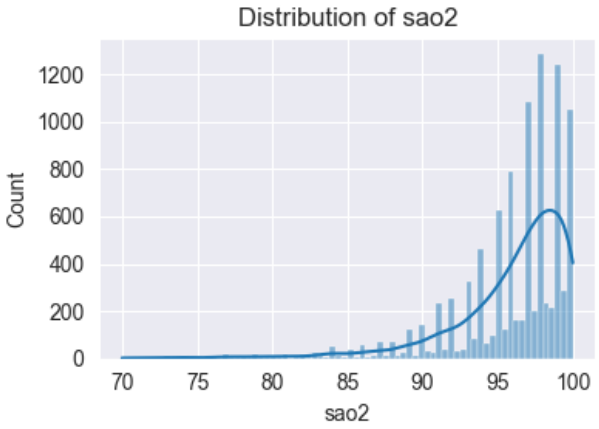


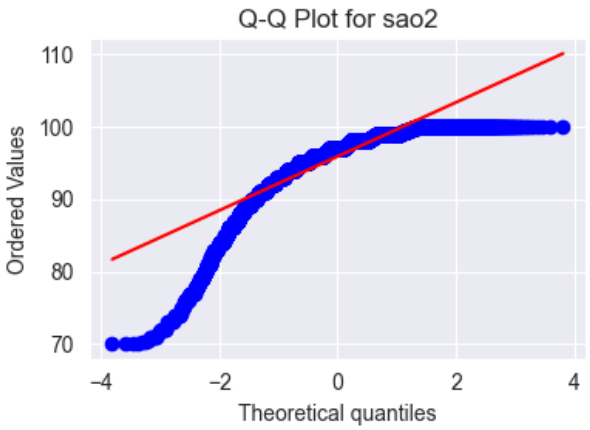

```
bmi_admission: Skewness = 70.7006
  Not normal

gap: Sampling 10,000 points from 49093 total
```



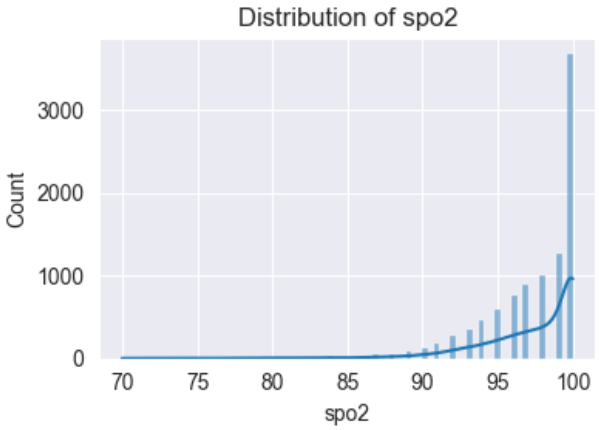


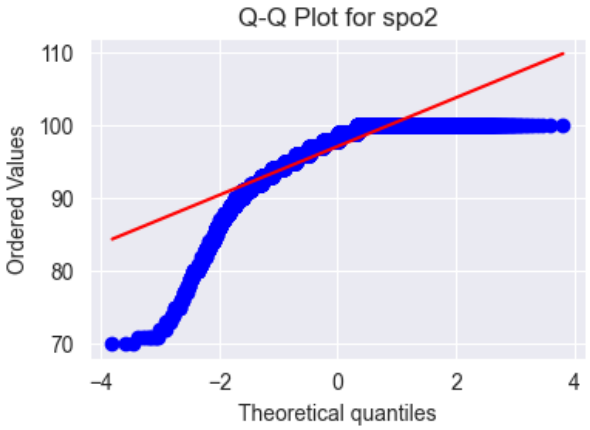

```
gap: Skewness = -0.1730
  Approximately normal

sao2: Sampling 10,000 points from 49093 total
```



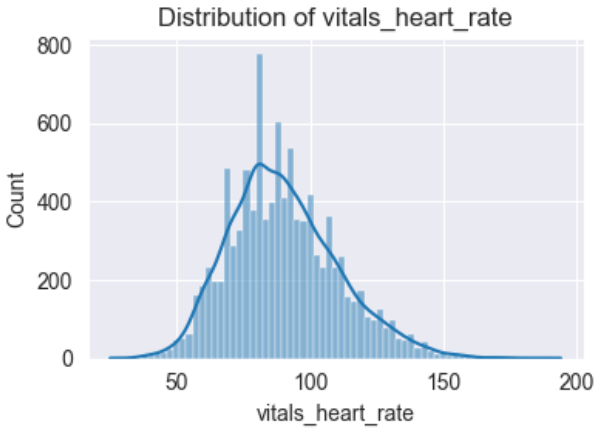


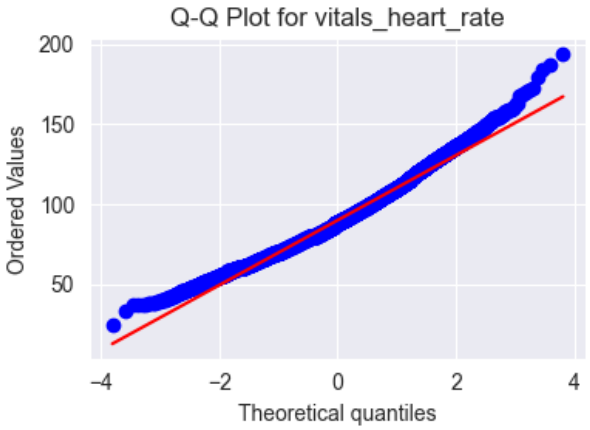

```
sao2: Skewness = -2.2332
  Not normal

spo2: Sampling 10,000 points from 49093 total
```



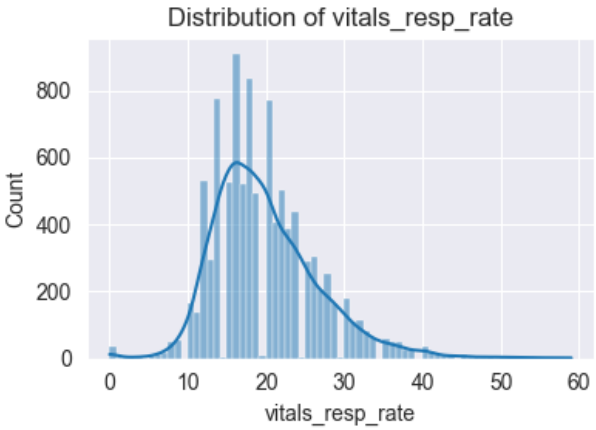


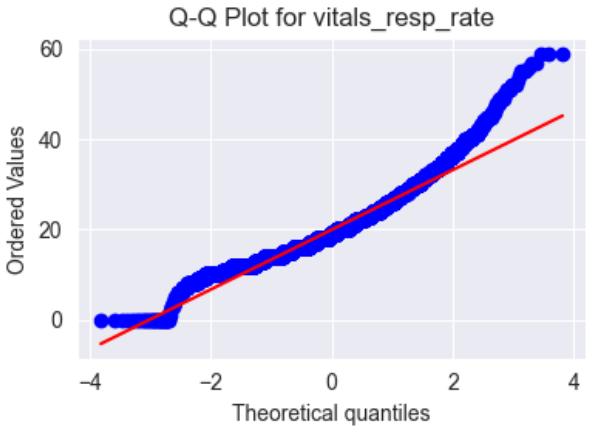

```
spo2: Skewness = -2.4713
  Not normal

vitals_heart_rate: Sampling 10,000 points from 43643 total
```






```
vitals_heart_rate: Skewness = 0.5217
  Not normal

vitals_resp_rate: Sampling 10,000 points from 42217 total
```






```
vitals_resp_rate: Skewness = 0.9852
  Not normal
```

Data (Im)Balance

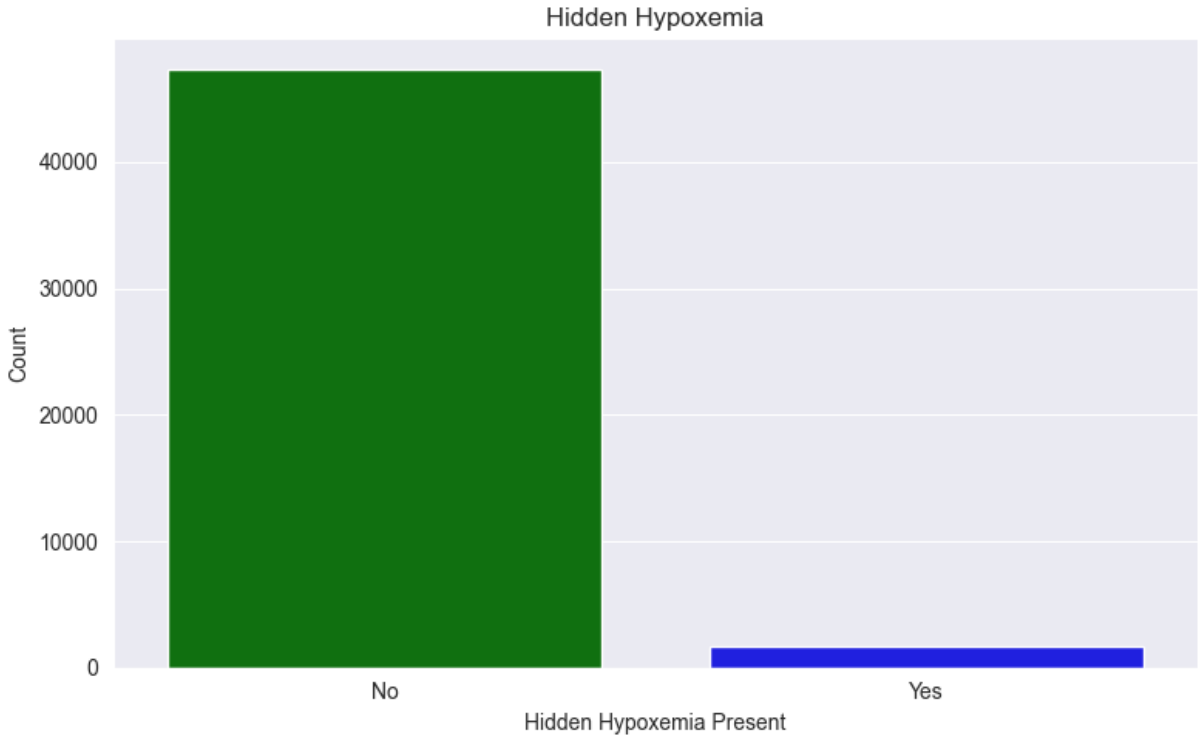
Check for imbalance in the data between the hidden hypoxemia and gap columns.

Return to C2: Data Preprocessing , Table of Contents

In [38]:
```
# Create a binary that identifies hidden hypoxemia cases
# Hidden hypoxemia: SpO2 >= 88% but SaO2 < 88%
df_hidden = ((df_lower['spo2'] >= 88) & (df_lower['sao2'] < 88)).astype(int)
# Check the distribution
print(f"Hidden hypoxemia cases: {df_hidden.sum()}")
print(f"Percentage: {df_hidden.mean() * 100:.2f}%")
```

```
Hidden hypoxemia cases: 1731
Percentage: 3.53%
```

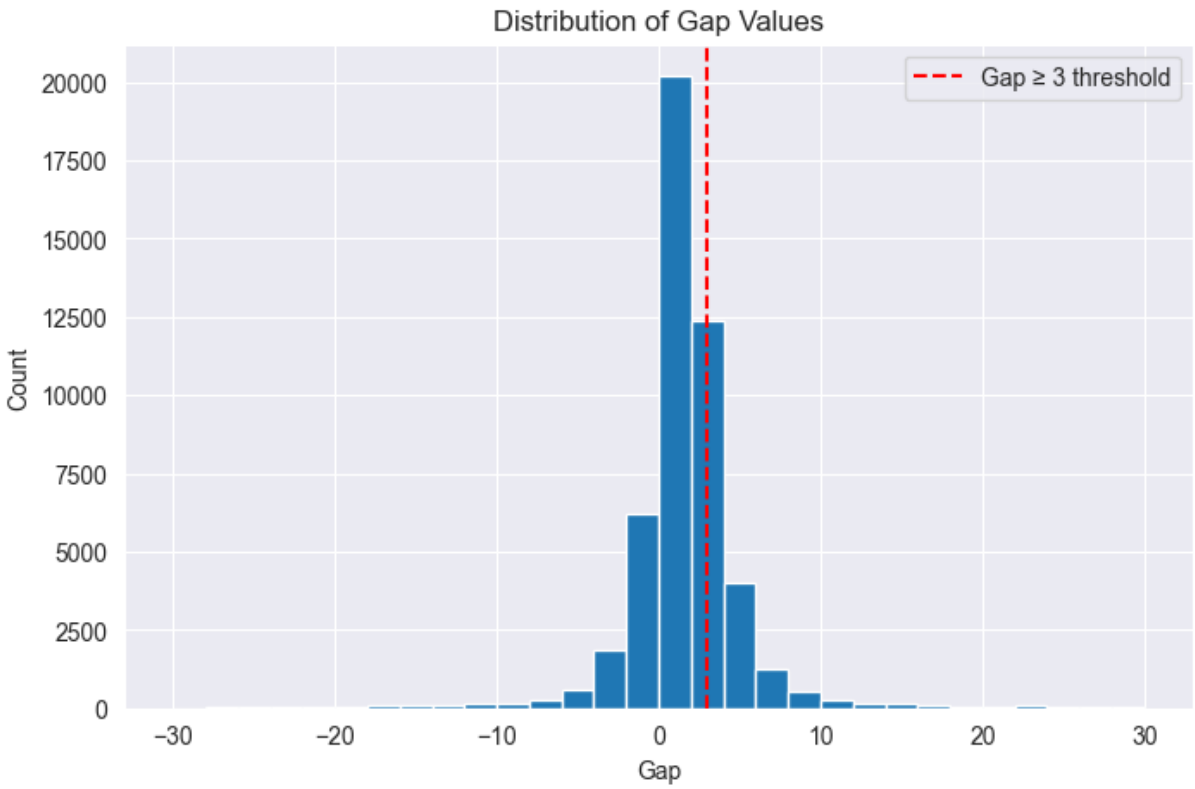
In [39]:
```
df_mapped = df_hidden.map({1: "Yes", 0: "No"})
plt.figure(figsize=(8, 5))
ax = sns.countplot(x=df_mapped, hue=df_mapped, palette=['green', 'blue'], le
plt.xlabel("Hidden Hypoxemia Present")
plt.ylabel("Count")
plt.title("Hidden Hypoxemia")
plt.tight_layout()
plt.show()
```



The gap threshold will be set at 3. This threshold was chosen because it helps identify patients with true hypoxemia (SaO2 ≤ 90%) even when their pulse oximetry readings appear normal (SpO2 ≥ 93%).

In [40]:
```
# Create a column for significant gap (≥ 3)
df_significant_gap = (df_lower['gap'] >= 3).astype(int)

# Check the distribution of this new column
print(f"Samples with gap ≥ 3: {df_significant_gap.sum()}")
print(f"Percentage: {df_significant_gap.mean() * 100:.2f}%")

# Visualize the distribution of the gap values with the new threshold
plt.figure(figsize=(8, 5))
plt.hist(df_lower['gap'], bins=30)
plt.axvline(x=3, color='r', linestyle='--', label='Gap ≥ 3 threshold')
plt.xlabel('Gap')
plt.ylabel('Count')
plt.title('Distribution of Gap Values')
plt.legend()
plt.show()
```

```
Samples with gap ≥ 3: 11457
Percentage: 23.34%
```



In [41]:
```
# Split the data using stratification
# Define features and target
features = df_lower.drop(['sao2', 'gap'], axis=1) #SaO2 is no longer needed
target = df_lower['gap']
stratification = df_significant_gap
#split the data using stratification
X_train, X_test, y_train, y_test = train_test_split(
    features,
    target,
    test_size=0.2,
    stratify=stratification,
    random_state=42
)
```

In [42]:
```
# Function to confirm alignment of data sets
def check_alignment(X_train, X_test, y_train, y_test):
    print(f"X_train: {X_train.shape}")
    print(f"y_train: {y_train.shape if hasattr(y_train, 'shape') else (len(y
    print(f"X_test: {X_test.shape}")
    print(f"y_test: {y_test.shape if hasattr(y_test, 'shape') else (len(y_te

    # Check if X and y match in rows
    if len(X_train) != len(y_train):
        print("ERROR: X_train and y_train row counts don't match!")
    if len(X_test) != len(y_test):
        print("ERROR: X_test and y_test row counts don't match!")

    if len(X_train) == len(y_train) and len(X_test) == len(y_test):
        print("All datasets aligned correctly!")

    return
```

In [43]:
```
check_alignment(X_train, X_test, y_train, y_test)
```

```
X_train: (39274, 34)
y_train: (39274,)
X_test: (9819, 34)
y_test: (9819,)
All datasets aligned correctly!
```

Outliers

Outliers were managed using the technique of winsorization. Any value that falls outside a prescribed percentile is replaced with the value at that percentile. This means that rather than cutting an outlier out of the data, it is recoded to the outside value (Horsch, 2021). This was important for this data set since many variables had non-error outliers, that is, the outliers were expected. Box plots, interquartile ranges, and percentile values were all used in the process of determining which values were to be kept and at which percentile. This process relied heavily on the subject matter expertise of the medical field.

Return to C2: Data Preprocessing , Table of Contents

In [44]:
```
#Function detection of outliers for Winsorization, using IQR
def winz_outliers(df):
    print("Outlier Analysis Report")
    print("=" * 50)
    for col in df:
        if pd.api.types.is_numeric_dtype(df[col]):
            # Check if column has NaN values
            nan_count = df[col].isna().sum()

            if nan_count == len(df):
                print(f"\nColumn: {col}")
                print("All values are NaN - skipping outlier analysis")
                continue
```

```
            col_data = df[col].dropna()

            if len(col_data) == 0:
                print(f"\nColumn: {col}")
                print("All values are NaN - skipping outlier analysis")
                continue

            Q1 = col_data.quantile(0.25)
            Q3 = col_data.quantile(0.75)
            IQR = Q3 - Q1
            outliers = ((col_data < (Q1 - 1.5 * IQR)) | (col_data > (Q3 + 1.
            outlier_count = outliers.sum()

            if outlier_count > 0:
                outer_fence = 3 * IQR
                outer_fence_low = Q1 - outer_fence
                outer_fence_up = Q3 + outer_fence

                print(f"\nColumn: {col}")
                print(f"Number of NaN values: {nan_count}")
                print(f"Number of Outliers: {outlier_count}")
                print(f"Minimum: {round(col_data.min(), 2)}")
                print(f"Lower Outer Fence: {round(outer_fence_low, 2)}")
                print(f"Maximum: {round(col_data.max(), 2)}")
                print(f"Upper Outer Fence: {round(outer_fence_up, 2)}")

    print("=" * 50)
```

In [45]:
```
# Examine outlier statistics
winz_outliers(X_train)
```

```
Outlier Analysis Report
==================================================

Column: admission_age
Number of NaN values: 1
Number of Outliers: 630
Minimum: 14.0
Lower Outer Fence: -8.0
Maximum: 90.0
Upper Outer Fence: 139.0

Column: weight_admission
Number of NaN values: 889
Number of Outliers: 1202
Minimum: 0.0
Lower Outer Fence: -27.3
Maximum: 771.2
Upper Outer Fence: 193.9

Column: height_admission
Number of NaN values: 1033
Number of Outliers: 268
Minimum: 0.0
Lower Outer Fence: 116.84
Maximum: 504.8
Upper Outer Fence: 223.52

Column: bmi_admission
Number of NaN values: 1586
Number of Outliers: 1561
Minimum: 0.0
Lower Outer Fence: -5.34
Maximum: 725551.02
Upper Outer Fence: 62.77

Column: spo2
Number of NaN values: 0
Number of Outliers: 1631
Minimum: 70.0
Lower Outer Fence: 84.0
Maximum: 100.0
Upper Outer Fence: 112.0

Column: vitals_heart_rate
Number of NaN values: 4380
Number of Outliers: 422
Minimum: 25.0
Lower Outer Fence: -6.0
Maximum: 214.0
Upper Outer Fence: 183.0

Column: vitals_resp_rate
Number of NaN values: 5496
Number of Outliers: 728
Minimum: 0.0
Lower Outer Fence: -12.0
```

```
Maximum: 66.0
Upper Outer Fence: 51.0

Column: vitals_mbp_ni
Number of NaN values: 12383
Number of Outliers: 537
Minimum: 14.0
Lower Outer Fence: -2.0
Maximum: 238.0
Upper Outer Fence: 159.0

Column: vitals_sbp_ni
Number of NaN values: 11955
Number of Outliers: 424
Minimum: 30.0
Lower Outer Fence: 1.0
Maximum: 250.0
Upper Outer Fence: 232.0

Column: vitals_tempc
Number of NaN values: 6360
Number of Outliers: 2374
Minimum: 23.7
Lower Outer Fence: 34.0
Maximum: 44.3
Upper Outer Fence: 39.6

Column: cbc_hemoglobin
Number of NaN values: 5396
Number of Outliers: 275
Minimum: 1.9
Lower Outer Fence: -0.5
Maximum: 96.0
Upper Outer Fence: 22.6

Column: cbc_hematocrit
Number of NaN values: 5338
Number of Outliers: 268
Minimum: 5.9
Lower Outer Fence: -1.5
Maximum: 72.0
Upper Outer Fence: 68.5

Column: cbc_mch
Number of NaN values: 8816
Number of Outliers: 1251
Minimum: 15.4
Lower Outer Fence: 19.8
Maximum: 48.8
Upper Outer Fence: 40.1

Column: cbc_mchc
Number of NaN values: 7382
Number of Outliers: 557
Minimum: 24.1
Lower Outer Fence: 25.6
```

Maximum: 48.8
Upper Outer Fence: 40.3

Column: cbc_mcv
Number of NaN values: 7376
Number of Outliers: 1017
Minimum: 58.1
Lower Outer Fence: 63.0
Maximum: 145.6
Upper Outer Fence: 119.0

Column: cbc_platelet
Number of NaN values: 5861
Number of Outliers: 1071
Minimum: 2.0
Lower Outer Fence: -226.0
Maximum: 1575.0
Upper Outer Fence: 614.0

Column: cbc_rbc
Number of NaN values: 6242
Number of Outliers: 249
Minimum: 0.66
Lower Outer Fence: -0.15
Maximum: 7.96
Upper Outer Fence: 7.62

Column: cbc_rdw
Number of NaN values: 8719
Number of Outliers: 1305
Minimum: 0.0
Lower Outer Fence: 5.6
Maximum: 58.6
Upper Outer Fence: 24.5

Column: cbc_wbc
Number of NaN values: 6143
Number of Outliers: 1282
Minimum: 0.05
Lower Outer Fence: -16.3
Maximum: 240.0
Upper Outer Fence: 41.8

Column: bmp_sodium
Number of NaN values: 5227
Number of Outliers: 1149
Minimum: 99.0
Lower Outer Fence: 118.0
Maximum: 188.0
Upper Outer Fence: 160.0

Column: bmp_potassium
Number of NaN values: 4747
Number of Outliers: 813
Minimum: 1.5
Lower Outer Fence: 1.0

Maximum: 9.9
Upper Outer Fence: 7.3

Column: bmp_chloride
Number of NaN values: 6308
Number of Outliers: 647
Minimum: 53.0
Lower Outer Fence: 73.0
Maximum: 160.0
Upper Outer Fence: 136.0

Column: bmp_bicarbonate
Number of NaN values: 8496
Number of Outliers: 1160
Minimum: 2.0
Lower Outer Fence: -1.0
Maximum: 63.0
Upper Outer Fence: 48.0

Column: bmp_bun
Number of NaN values: 6188
Number of Outliers: 2294
Minimum: 1.0
Lower Outer Fence: -49.0
Maximum: 259.0
Upper Outer Fence: 98.0

Column: bmp_creatinine
Number of NaN values: 6211
Number of Outliers: 3279
Minimum: 0.1
Lower Outer Fence: -2.08
Maximum: 26.6
Upper Outer Fence: 4.64

Column: bmp_glucose
Number of NaN values: 3762
Number of Outliers: 2069
Minimum: 17.0
Lower Outer Fence: -85.0
Maximum: 1485.0
Upper Outer Fence: 370.0

Column: bmp_aniongap
Number of NaN values: 11007
Number of Outliers: 595
Minimum: -2.0
Lower Outer Fence: -13.0
Maximum: 62.0
Upper Outer Fence: 36.0

Column: bmp_calcium
Number of NaN values: 8707
Number of Outliers: 1268
Minimum: 0.63
Lower Outer Fence: 4.5

```
Maximum: 22.2
Upper Outer Fence: 12.2

Column: hfp_alt
Number of NaN values: 14091
Number of Outliers: 3093
Minimum: 1.0
Lower Outer Fence: -76.0
Maximum: 12456.0
Upper Outer Fence: 141.0

Column: hfp_alp
Number of NaN values: 14296
Number of Outliers: 1776
Minimum: 7.0
Lower Outer Fence: -96.0
Maximum: 2820.0
Upper Outer Fence: 275.0

Column: hfp_ast
Number of NaN values: 13978
Number of Outliers: 3354
Minimum: 4.0
Lower Outer Fence: -109.0
Maximum: 42606.0
Upper Outer Fence: 192.0

Column: hfp_albumin
Number of NaN values: 14149
Number of Outliers: 44
Minimum: 0.6
Lower Outer Fence: -0.8
Maximum: 6.3
Upper Outer Fence: 6.9
==================================================
```

Values to winsorize:

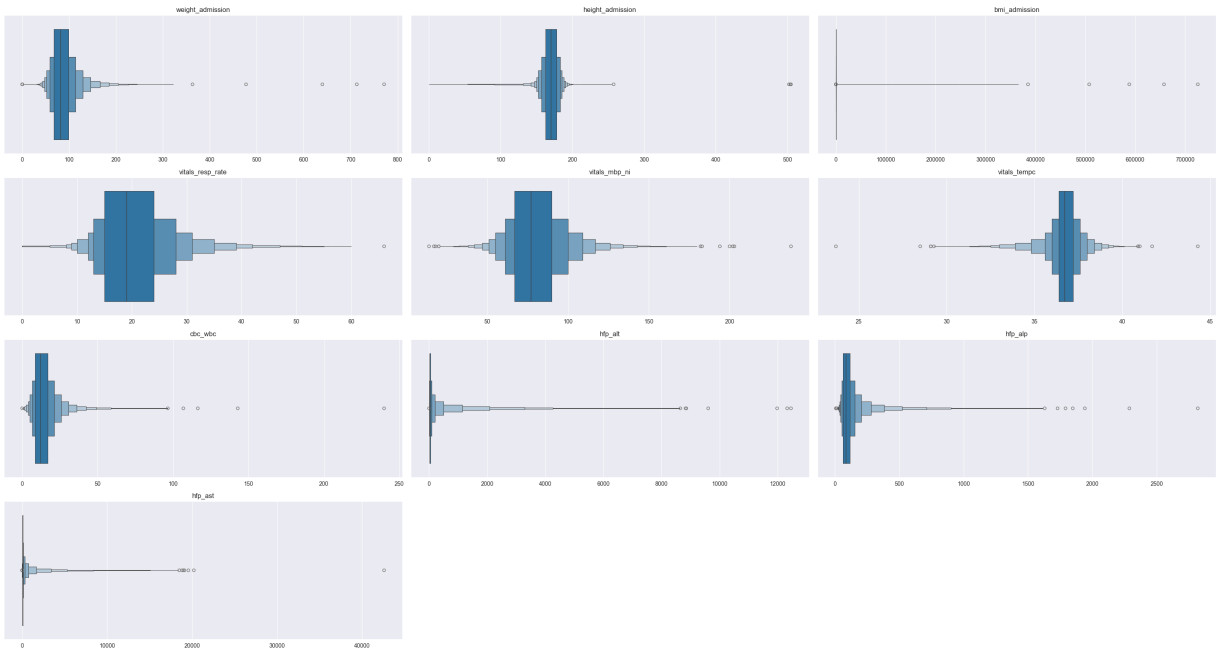
- weight — upper and lower
- height — upper and lower
- bmi - upper and lower
- Resp_rate — lower
- mean bp — upper and lower
- temp — lower
- wbc - upper
- alt - upper and lower
- alp - upper and lower
- ast - upper and lower

In [46]:
```python
# Function to create boxplots
def box_plot_viz(df, figsize=(10, 4), n_cols=3):
    n_plots = len(df.columns)
    n_rows = (n_plots + n_cols - 1) // n_cols
```

```
    fig, axes = plt.subplots(n_rows, n_cols, figsize=(figsize[0]*n_cols, fig
    axes = axes.flatten() if n_plots > 1 else [axes]

    for i, column in enumerate(df.columns):
        sns.boxenplot(x=df[column], ax=axes[i])
        axes[i].set_title(column)
        axes[i].set_xlabel('')

    for j in range(i+1, len(axes)):
        axes[j].set_visible(False)

    plt.tight_layout()
    plt.show()
```

Variables that had unexpected values as determined by the outlier function were visualized.

In [47]:
```
# Create a boxplot for each column that has unexpected values
outlier_shape = X_train[['weight_admission', 'height_admission', 'bmi_admiss
box_plot_viz(outlier_shape)
```



A function was created to examine the percentiles of the variables identified as having outliers. This allowed for the determination of non-error outlier values versus outliers that were due to errors. Each identified variable was examined individually to set custom percentile ranges for the winsorization.

In [48]:
```
# Function to find winsorization markers
def print_quantiles(column):
    # Lower quantiles
    print('Lower quantiles:')
    print('0.5% quantile:  ', column.quantile(0.005))
    print('1% quantile:    ', column.quantile(0.01))
    print('5% quantile:    ', column.quantile(0.05))
    print('10% quantile:   ', column.quantile(0.10))
    # Upper Quantiles
```

```
    print('\nUpper quantiles:')
    print('90% quantile:   ', column.quantile(0.90))
    print('92.5% quantile: ', column.quantile(0.925))
    print('95% quantile:   ', column.quantile(0.95))
    print('97.5% quantile: ', column.quantile(0.975))
    print('99% quantile:   ', column.quantile(0.99))
    print('99.9% quantile: ', column.quantile(0.999))
```

In [49]:
```
print_quantiles(X_train['weight_admission'])
```

```
Lower quantiles:
0.5% quantile:   38.092000000000006
1% quantile:     40.8
5% quantile:     50.3
10% quantile:    56.47

Upper quantiles:
90% quantile:    118.8
92.5% quantile:  125.0
95% quantile:    135.0
97.5% quantile:  152.6
99% quantile:    178.9
99.9% quantile:  244.4677600000002
```

In [50]:
```
print_quantiles(X_train['height_admission'])
```

```
Lower quantiles:
0.5% quantile:   136.20000000000002
1% quantile:     144.7
5% quantile:     152.4
10% quantile:    155.0

Upper quantiles:
90% quantile:    182.9
92.5% quantile:  185.0
95% quantile:    185.42
97.5% quantile:  188.0
99% quantile:    193.0
99.9% quantile:  200.7
```

In [51]:
```
print_quantiles(X_train['bmi_admission'])
```

```
Lower quantiles:
0.5% quantile:   14.69
1% quantile:     15.82
5% quantile:     18.91
10% quantile:    20.67

Upper quantiles:
90% quantile:    40.50300000000003
92.5% quantile:  42.95474999999999
95% quantile:    46.32
97.5% quantile:  53.14824999999997
99% quantile:    63.32649999999987
99.9% quantile:  266.92186000000237
```

In [52]:
```
print_quantiles(X_train['vitals_resp_rate'])
```

```
Lower quantiles:
0.5% quantile:   6.0
1% quantile:     8.0
5% quantile:     12.0
10% quantile:    12.0

Upper quantiles:
90% quantile:    29.0
92.5% quantile:  30.0
95% quantile:    32.0
97.5% quantile:  36.0
99% quantile:    41.0
99.9% quantile:  54.22300000000541
```

In [53]:
```
print_quantiles(X_train['vitals_mbp_ni'])
```

```
Lower quantiles:
0.5% quantile:   39.0
1% quantile:     44.0
5% quantile:     54.0
10% quantile:    59.0

Upper quantiles:
90% quantile:    103.0
92.5% quantile:  107.0
95% quantile:    111.0
97.5% quantile:  120.0
99% quantile:    131.0
99.9% quantile:  160.11000000000422
```

In [54]:
```
print_quantiles(X_train['vitals_tempc'])
```

```
Lower quantiles:
0.5% quantile:   32.7
1% quantile:     33.3
5% quantile:     35.39
10% quantile:    35.9

Upper quantiles:
90% quantile:    37.8
92.5% quantile:  37.9
95% quantile:    38.17
97.5% quantile:  38.6
99% quantile:    39.1
99.9% quantile:  40.1
```

In [55]:
```
print_quantiles(X_train['cbc_wbc'])
```

```
Lower quantiles:
0.5% quantile:   1.1
1% quantile:     2.0
5% quantile:     4.7
10% quantile:    6.1

Upper quantiles:
90% quantile:    22.6
92.5% quantile:  24.6
95% quantile:    27.6
97.5% quantile:  32.5
99% quantile:    39.92699999999997
99.9% quantile:  72.1348000000001
```

In [56]:
```
print_quantiles(X_train['hfp_alt'])
```

```
Lower quantiles:
0.5% quantile:   6.0
1% quantile:     6.0
5% quantile:     9.0
10% quantile:    12.0

Upper quantiles:
90% quantile:    118.0
92.5% quantile:  162.35000000000218
95% quantile:    271.0
97.5% quantile:  659.0
99% quantile:    1701.7000000000044
99.9% quantile:  5541.992000000129
```

In [57]:
```
print_quantiles(X_train['hfp_alp'])
```

```
Lower quantiles:
0.5% quantile:   26.0
1% quantile:     30.0
5% quantile:     42.0
10% quantile:    49.0

Upper quantiles:
90% quantile:    170.0
92.5% quantile:  191.0
95% quantile:    225.0
97.5% quantile:  309.0
99% quantile:    473.0
99.9% quantile:  1130.6670000000304
```

In [58]:
```
print_quantiles(X_train['hfp_ast'])
```

```
Lower quantiles:
0.5% quantile:   8.0
1% quantile:     9.0
5% quantile:     12.0
10% quantile:    15.0

Upper quantiles:
90% quantile:    173.0
92.5% quantile:  245.0
95% quantile:    429.0
97.5% quantile:  1014.625
99% quantile:    2735.249999999982
99.9% quantile:  10722.73500000099
```

Values to winsorize:

- weight — upper and lower (.01, 0.001)
- height — upper and lower (0.005, 0.001)
- bmi - upper and lower (0.05, 0.01)
- Resp_rate - lower (0.005, 0)
- mean bp- upper and lower (0.005, 0.001)
- temp - lower (0.005, 0)
- wbc - upper (0, 0.001)
- alt - upper and lower (0, 0.025)
- alp- upper and lower (0, 0.01)
- ast - upper and lower (0, 0.05)

In [59]:
```
## Winsorize the outliers
import warnings
warnings.filterwarnings('ignore', message="Warning: 'partition' will ignore
#When using the limits, you are specifying "how much" to take off, not the "
X_train['weight_winz'] = winsorize(X_train['weight_admission'].values, limit
X_train['height_winz'] = winsorize(X_train['height_admission'].values, limit
X_train['bmi_winz'] = winsorize(X_train['bmi_admission'].values, limits=(0.0
X_train['resp_winz'] = winsorize(X_train['vitals_resp_rate'].values, limits=
X_train['mbp_winz'] = winsorize(X_train['vitals_mbp_ni'].values, limits=(0.0
X_train['temp_winz'] = winsorize(X_train['vitals_tempc'].values, limits=(0.0
X_train['wbc_winz'] = winsorize(X_train['cbc_wbc'].values, limits=(0, 0.001)
X_train['alt_winz'] = winsorize(X_train['hfp_alt'].values, limits=(0, 0.05))
X_train['alp_winz'] = winsorize(X_train['hfp_alp'].values, limits=(0, 0.01))
X_train['ast_winz'] = winsorize(X_train['hfp_ast'].values, limits=(0, 0.005)
```

In [60]:
```
#Verify the addition of the winsorized columns
X_train.tail(10)
```

Out[60]:

	admission_age	sex_female	weight_admission	height_admission	bmi_admissi
1578	52.0	0	50.6	175.26	16
43950	56.0	0	131.0	185.42	38
12757	67.0	1	98.3	160.00	38.
36878	87.0	0	85.0	180.40	26
28758	64.0	1	89.4	167.60	31.
10110	54.0	0	100.0	182.00	30
42845	69.0	1	77.1	157.50	31.
32826	78.0	0	68.7	170.00	23
31704	74.0	1	98.2	172.70	32.
18863	72.0	1	122.0	157.00	49.

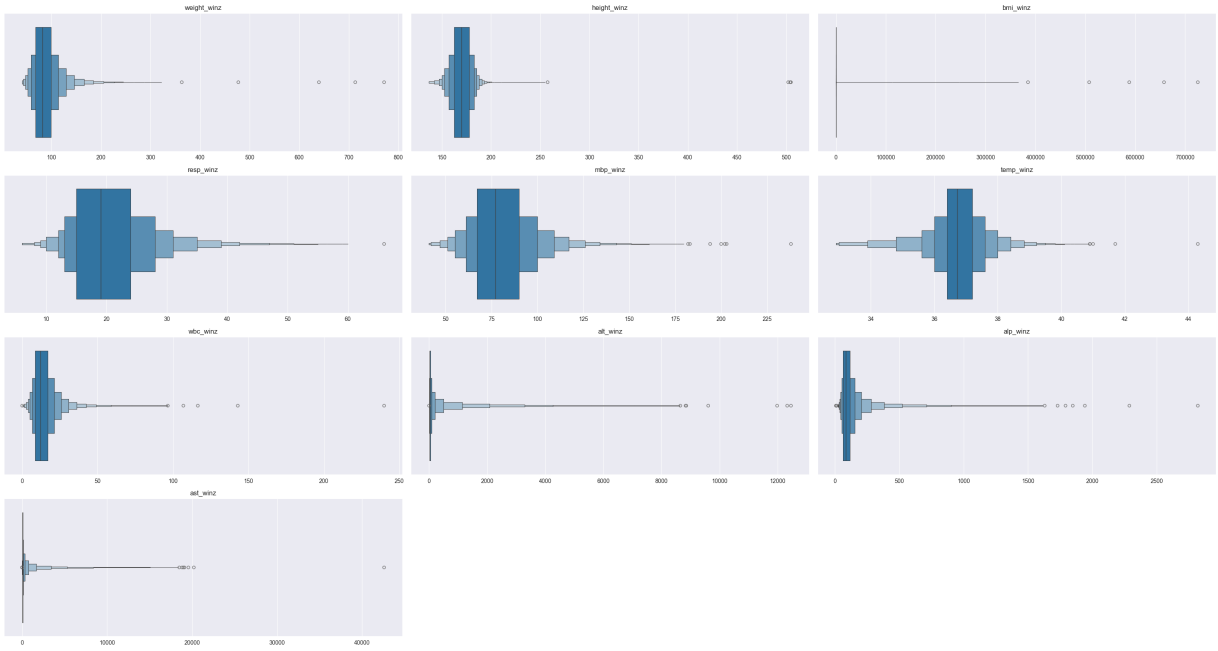
In [61]:
```
X_train.describe()
```

Out[61]:

	admission_age	sex_female	weight_admission	height_admission	bmi_admis
count	39273.000000	39274.000000	38385.000000	38241.000000	37688.000
mean	64.398238	0.436447	85.733947	169.386713	197.634
std	15.807949	0.495951	27.951675	12.868888	8270.32
min	14.000000	0.000000	0.000000	0.000000	0.000
25%	55.000000	0.000000	67.500000	162.560000	23.850
50%	66.000000	0.000000	81.600000	170.000000	28.080
75%	76.000000	1.000000	99.100000	177.800000	33.580
max	90.000000	1.000000	771.200000	504.800000	725551.020

The newly created columns were visualized using the box plot function.

In [62]:
```
# Examine the new winsorized columns
winz_df = X_train[['weight_winz', 'height_winz', 'bmi_winz', 'resp_winz', 'm
box_plot_viz(winz_df)
```



In [63]:
```
#Remake data set with the outliers columns removed
X_train_winz = X_train.drop(columns=outlier_shape, axis = 1)
#verify removal
print(X_train_winz.columns)
```

```
Index(['admission_age', 'sex_female', 'race_ethnicity', 'spo2',
       'vitals_heart_rate', 'vitals_sbp_ni', 'cbc_hemoglobin',
       'cbc_hematocrit', 'cbc_mch', 'cbc_mchc', 'cbc_mcv', 'cbc_platelet',
       'cbc_rbc', 'cbc_rdw', 'bmp_sodium', 'bmp_potassium', 'bmp_chloride',
       'bmp_bicarbonate', 'bmp_bun', 'bmp_creatinine', 'bmp_glucose',
       'bmp_aniongap', 'bmp_calcium', 'hfp_albumin', 'weight_winz',
       'height_winz', 'bmi_winz', 'resp_winz', 'mbp_winz', 'temp_winz',
       'wbc_winz', 'alt_winz', 'alp_winz', 'ast_winz'],
      dtype='object')
```

The winsorization was then performed on the test data. Test data was not used in the process of determining which variables or percentiles to treat in regard to outliers. By keeping the test data outside the evaluation of outliers, data leakage was prevented.

In [64]:
```
# Apply training winsorization limits to the test data
X_test['weight_winz'] = winsorize(X_test['weight_admission'].values, limits=
X_test['height_winz'] = winsorize(X_test['height_admission'].values, limits=
X_test['bmi_winz'] = winsorize(X_test['bmi_admission'].values, limits=(0.05,
X_test['resp_winz'] = winsorize(X_test['vitals_resp_rate'].values, limits=(0
X_test['mbp_winz'] = winsorize(X_test['vitals_mbp_ni'].values, limits=(0.005
X_test['temp_winz'] = winsorize(X_test['vitals_tempc'].values, limits=(0.005
X_test['wbc_winz'] = winsorize(X_test['cbc_wbc'].values, limits=(0, 0.001))
X_test['alt_winz'] = winsorize(X_test['hfp_alt'].values, limits=(0, 0.05))
X_test['alp_winz'] = winsorize(X_test['hfp_alp'].values, limits=(0, 0.01))
X_test['ast_winz'] = winsorize(X_test['hfp_ast'].values, limits=(0, 0.005))
```

In [65]:
```
#Remake data set with the outliers removed
X_test_winz = X_test.drop(columns=outlier_shape, axis = 1)
#verify removal
print(X_test_winz.columns)
```

```
Index(['admission_age', 'sex_female', 'race_ethnicity', 'spo2',
       'vitals_heart_rate', 'vitals_sbp_ni', 'cbc_hemoglobin',
       'cbc_hematocrit', 'cbc_mch', 'cbc_mchc', 'cbc_mcv', 'cbc_platelet',
       'cbc_rbc', 'cbc_rdw', 'bmp_sodium', 'bmp_potassium', 'bmp_chloride',
       'bmp_bicarbonate', 'bmp_bun', 'bmp_creatinine', 'bmp_glucose',
       'bmp_aniongap', 'bmp_calcium', 'hfp_albumin', 'weight_winz',
       'height_winz', 'bmi_winz', 'resp_winz', 'mbp_winz', 'temp_winz',
       'wbc_winz', 'alt_winz', 'alp_winz', 'ast_winz'],
      dtype='object')
```

In [66]:
```
check_alignment(X_train_winz, X_test_winz, y_train, y_test)
```

```
X_train: (39274, 34)
y_train: (39274,)
X_test: (9819, 34)
y_test: (9819,)
All datasets aligned correctly!
```

In [67]:
```
# Function to check sparsity
def sparcity_check(X_train, X_test, y_train, y_test):   # the function is sp
    if not isinstance(y_train, pd.DataFrame):
        y_train = pd.DataFrame(y_train, columns=['target'])
    if not isinstance(y_test, pd.DataFrame):
        y_test = pd.DataFrame(y_test, columns=['target'])
    train_full = pd.concat([X_train, y_train], axis=1)
    test_full = pd.concat([X_test, y_test], axis=1)

    # Combine train and test
    combined_df = pd.concat([train_full, test_full], axis=0)
    total_nulls = combined_df.isna().sum().sum()
    total_cells = combined_df.size
    sparsity_percentage = round(total_nulls/total_cells*100, 2)

    print(f"Total sparsity: {total_nulls} missing values out of {total_cells

    return sparsity_percentage
```

Clean Data Sparsity

In [68]:
```
sparcity_check(X_train, X_test, y_train, y_test)
```

```
Total sparsity: 423026 missing values out of 2209185 cells (19.15%)
```

Out[68]: 19.15

Encode Categorical Variables

Categorical variables must be encoded into numerical values for the RFR model

---

Return to C2: Data Preprocessing , Table of Contents

In [69]:
```
#Function to encode the data - one-hot encoding (5 or less) and target encod
def wrangle_cat(df, target_values=None, target_col=None, max_categories_for_
    result_df = df.copy()
```

```
    for col in result_df.columns:
        if pd.api.types.is_string_dtype(result_df[col]):
            result_df[col] = result_df[col].astype(str).str.strip().str.lowe
            if result_df[col].isin(['yes', 'no', 'true', 'false']).all():
                mapping = {'yes': 1, 'no': 0, 'true': 1, 'false': 0}
                result_df[col] = result_df[col].map(mapping)

    categorical_cols = result_df.columns[result_df.dtypes == 'object']
    high_cardinality_cols = []
    low_cardinality_cols = []
    encoding_maps = {}

    for col in categorical_cols:
        if result_df[col].nunique() > max_categories_for_onehot:
            high_cardinality_cols.append(col)
        else:
            low_cardinality_cols.append(col)

    for col in high_cardinality_cols:
        if is_train:
            df_with_target = pd.DataFrame({col: df[col], 'target': target_va
            target_means = df_with_target.groupby(col)['target'].mean()
            encoding_maps[col] = target_means

            result_df[col + '_target_encoded'] = result_df[col].map(target_m
        else:
            result_df[col + '_target_encoded'] = result_df[col].map(target_c

        result_df = result_df.drop(col, axis=1)

    if low_cardinality_cols:
        result_df = pd.get_dummies(result_df, columns=low_cardinality_cols,

    if is_train:
        return result_df, encoding_maps
    else:
        return result_df
```

Training and test data were encoded separately to prevent data leakage.

In [70]:
```
# For training data
X_train_encode, encoding_maps = wrangle_cat(X_train_winz, target_values=y_tr
X_train_encode.head(10)
```

Out[70]:

	admission_age	sex_female	spo2	vitals_heart_rate	vitals_sbp_ni	cbc_hemo
17744	77.0	0	96.0	NaN	NaN	
36775	90.0	1	86.0	NaN	NaN	
29773	67.0	1	100.0	89.0	161.0	
521	86.0	1	99.0	66.0	NaN	
30702	19.0	0	97.0	114.0	109.0	
17508	58.0	0	97.0	NaN	NaN	
21502	85.0	0	100.0	98.0	88.0	
35909	74.0	0	99.0	NaN	NaN	
47211	22.0	1	94.0	113.0	119.0	
39133	48.0	0	99.0	110.0	112.0	

In [71]:
```
# For test data
X_test_encode = wrangle_cat(X_test_winz, target_col=encoding_maps, is_train=
X_test_encode.head(10)
```

Out[71]:

	admission_age	sex_female	spo2	vitals_heart_rate	vitals_sbp_ni	cbc_hemo
38120	83.0	0	100.0	133.0	89.0	
9312	61.0	0	94.0	82.0	72.0	
8928	85.0	1	97.0	78.0	NaN	
1827	71.0	1	100.0	74.0	151.0	
9513	89.0	1	86.0	67.0	119.0	
30552	31.0	1	96.0	81.0	112.0	
41312	68.0	1	94.0	66.0	111.0	
3500	67.0	0	100.0	120.0	146.0	
33455	90.0	0	92.0	NaN	NaN	
11554	25.0	0	100.0	95.0	101.0	

Imputation of the Data

Impute the *Nans* using the Bayesian Ridge iterative imputer

Return to C2: Data Preprocessing , Table of Contents

In [72]:
```
# Handle NaNs with imputation
# BayesianRidge as estimator
imputer = IterativeImputer(
    estimator=BayesianRidge(),
```

```
    n_nearest_features=10
)
imputer.fit(X_train_encode)
X_train_imputed = imputer.transform(X_train_encode)
X_test_imputed = imputer.transform(X_test_encode)
```

In [73]:
```
# Convert imputed NumPy arrays back to pandas DataFrames
X_train_imputed_df = pd.DataFrame(X_train_imputed, columns=X_train_encode.co
X_test_imputed_df = pd.DataFrame(X_test_imputed, columns=X_test_encode.colum
```

In [74]:
```
# Reset indices on both dataframes
X_train_imputed_df = X_train_imputed_df.reset_index(drop=True)
y_train = y_train.reset_index(drop=True)

#create the combined dataframe
train_data = X_train_imputed_df.copy()
train_data['gap'] = y_train

# Verify the indices are aligned
print(f"X_train_encoded shape: {X_train_encode.shape}")
print(f"y_train shape: {y_train.shape}")
print(f"train_data shape: {train_data.shape}")
print(f"'gap' in train_data columns: {'gap' in train_data.columns}")
```

```
X_train_encoded shape: (39274, 34)
y_train shape: (39274,)
train_data shape: (39274, 35)
'gap' in train_data columns: True
```

Feature Engineering

Several features were added to the data since they were not in the original data set. These features are :

- BUN/ Creatinine ratio: a measurement of how well the kidneys are functioning. Elevated ratios may indicate dehydration or kidney problems.
- Oxygen Carrying capacity: how much oxygen a person is able to deliver to the body, based on hemoglobin levels.
- Predicted body weight: An estimate based on height and gender used to determine appropriate ventilator settings. This value serves as a surrogate for lung size.

Return to B: Data Collection , Table of Contents

In [75]:
```
# will add column for o2 carrying capacity
X_train_imputed_df.insert(10, 'o2_carry', X_train_imputed_df['cbc_hemoglobin
X_test_imputed_df.insert(10, 'o2_carry', X_test_imputed_df['cbc_hemoglobin']
X_train_imputed_df.head(10)
```

Out[75]:

	admission_age	sex_female	spo2	vitals_heart_rate	vitals_sbp_ni	cbc_hemoglobir
0	77.0	0.0	96.0	93.651957	115.399476	13.100000
1	90.0	1.0	86.0	88.343687	109.792537	10.962366
2	67.0	1.0	100.0	89.000000	161.000000	7.000000
3	86.0	1.0	99.0	66.000000	106.602061	11.000000
4	19.0	0.0	97.0	114.000000	109.000000	11.662283
5	58.0	0.0	97.0	99.874132	117.383901	8.800000
6	85.0	0.0	100.0	98.000000	88.000000	9.500000
7	74.0	0.0	99.0	70.602423	111.919759	11.246928
8	22.0	1.0	94.0	113.000000	119.000000	9.800000
9	48.0	0.0	99.0	110.000000	112.000000	15.000000

In [76]:
```
# Add BUN/ Creatinine ratio
X_train_imputed_df.insert(25, 'bun_cr_ratio', X_train_imputed_df['bmp_bun']
X_test_imputed_df.insert(25, 'bun_cr_ratio', X_test_imputed_df['bmp_bun'] /
X_test_imputed_df.head(10)
```

Out[76]:

	admission_age	sex_female	spo2	vitals_heart_rate	vitals_sbp_ni	cbc_hemoglobir
0	83.0	0.0	100.0	133.000000	89.000000	8.0
1	61.0	0.0	94.0	82.000000	72.000000	8.7
2	85.0	1.0	97.0	78.000000	114.351516	10.0
3	71.0	1.0	100.0	74.000000	151.000000	7.7
4	89.0	1.0	86.0	67.000000	119.000000	12.4
5	31.0	1.0	96.0	81.000000	112.000000	7.1
6	68.0	1.0	94.0	66.000000	111.000000	14.6
7	67.0	0.0	100.0	120.000000	146.000000	8.4
8	90.0	0.0	92.0	83.042568	114.388373	10.7
9	25.0	0.0	100.0	95.000000	101.000000	14.8

In [77]:
```
# Will add a predicted body weight column, as that is what ventilator settin
X_train_imputed_df.insert(32, 'pbw', X_train_imputed_df.apply(lambda x:
    (45.5 + 0.91 * (x['height_winz'] - 152.4)) if x['sex_female'] == 1
    else (50 + 0.91 * (x['height_winz'] - 152.4)), axis=1))

X_test_imputed_df.insert(32, 'pbw', X_test_imputed_df.apply(lambda x:
    (45.5 + 0.91 * (x['height_winz'] - 152.4)) if x['sex_female'] == 1
    else (50 + 0.91 * (x['height_winz'] - 152.4)), axis=1))

X_train_imputed_df.head(10)
```

Out[77]:

	admission_age	sex_female	spo2	vitals_heart_rate	vitals_sbp_ni	cbc_hemoglobir
0	77.0	0.0	96.0	93.651957	115.399476	13.100000
1	90.0	1.0	86.0	88.343687	109.792537	10.962366
2	67.0	1.0	100.0	89.000000	161.000000	7.000000
3	86.0	1.0	99.0	66.000000	106.602061	11.000000
4	19.0	0.0	97.0	114.000000	109.000000	11.662283
5	58.0	0.0	97.0	99.874132	117.383901	8.800000
6	85.0	0.0	100.0	98.000000	88.000000	9.500000
7	74.0	0.0	99.0	70.602423	111.919759	11.246928
8	22.0	1.0	94.0	113.000000	119.000000	9.800000
9	48.0	0.0	99.0	110.000000	112.000000	15.000000

In [78]:
```
X_train_imputed_df.shape
```

Out[78]:
```
(39274, 37)
```

In [79]:
```
check_alignment(X_train_imputed_df, X_test_imputed_df, y_train, y_test)
```

```
X_train: (39274, 37)
y_train: (39274,)
X_test: (9819, 37)
y_test: (9819,)
All datasets aligned correctly!
```

In [80]:
```
sparcity_check(X_train_imputed_df, X_test_imputed_df, y_train, y_test)
```

```
Total sparsity: 49093 missing values out of 1865534 cells (2.63%)
```

Out[80]:
```
2.63
```

D: Random Forest Regression Analysis

This analysis created two different models, one with the synthesized data to handle the data imbalance, and one that only used the stratified data. The SMOGN model had worse results than the stratified only data and required significantly more computational time that included six hours for balancing of the data and over ten hours for the hyperparameter tuning. Further tuning of the parameters within the SMOTER or Optuna was prohibitive due to these extended run times. The stratified only data was used for the final model and analysis. The SMOTER model creation and results can be found in the Appendix.

After the data was cleaned, prepared and split, it was then used to create a random forest regression (RFR) model. RFR was chosen for its ability to handle a large number of variables. It is an ensemble method, meaning that it creates many different trees and

averages the results to reduce variability. An attribute of an RFR model is the ability to determine which features are the most important in regard to the model's accuracy. It is more resistant to overfitting, but it can be more difficult to interpret than other types of regression models.

An initial model was built and baseline metrics calculated. The metrics used to evaluate the model were:

- Mean absolute error (MAE) : The average absolute measure of the difference between predicted and actual values. The units are the same as the target variable. Lower values indicate better model performance.

  > $MAE = (1/n) * \Sigma|y_i - \hat{y}_i|$

- Mean squared error (MSE) : The average measure of predicted and actual values, squared. Larger errors are penalized more due to the values being squared, and lower values indicate better model performance.

  > $MSE = (1/n) * \Sigma(y_i - \hat{y}_i)^2$

- Root mean squared error (RMSE): This is the root value of MSE and is in the same units as the target variable. Larger errors are penalized, and it is easier to interpret than MSE. Lower values are again indicative of better model performance.

  > $RMSE = \sqrt{[(1/n) * \Sigma(y_i - \hat{y}_i)^2]} = \sqrt{MSE}$

- R2 score (Coefficient of Determination): The measurement of the amount variance in the target variable that can be predicted from the independent variables. This typically ranges from 0 to 1, with higher values representing more robust models. Values less than zero indicate that a model's performance is worse than if the mean had simply been used as the prediction.

  > $R^2 = 1 - [\Sigma(y_i - \hat{y}_i)^2 / \Sigma(y_i - \bar{y})^2] = 1 - (MSE / \text{Variance of } y)$

After the basic model was evaluated, feature selection was performed. This is done to reduce dimensionality and improve model performance. A disadvantage to this technique of using only the most important variables is that some features may be removed that have a complex relationship with the target variable.

An Optuna study was performed for hyperparameter optimization (Preferred Networks, Inc., 2024). Optuna is able to search for the best hyperparameters, as well as ending poor performing trials early. This allows for reduced computational time. A weakness of the Optuna studies is that multiple studies may need to be done in order to ensure that the optimal hyperparameters have been found.

The final model was trained using the Optuna hyperparameters. The test data was then used in the model and the evaluation metrics compared with those of the baseline

model. While there was some improvement in metrics with the final model, overall, the improvement was not enough to reject the stated null hypothesis that a random forest model cannot detect the difference between the SpO2 and the SaO2 with an RMSE of less than 3 percentage points.

Visualizations for stakeholders were created using the package SHapley Additive exPlanations (SHAP), which aims to simplify the explanation of machine learning models results. The visualization allows for an easier understanding of the impact the features have on the model. The SHAP values presented are based on the game theory developed by Lloyd S. Shapley in 1953 that assigns values to the features based on the contribution to the prediction (Molnar, 2025). An advantage of this package is the built-in visualizations and computations. A shortcoming is that the results may be misunderstood to be the causal relationship the features have with the target variable rather than the influence on the model.

Initial Random Forest Regression model

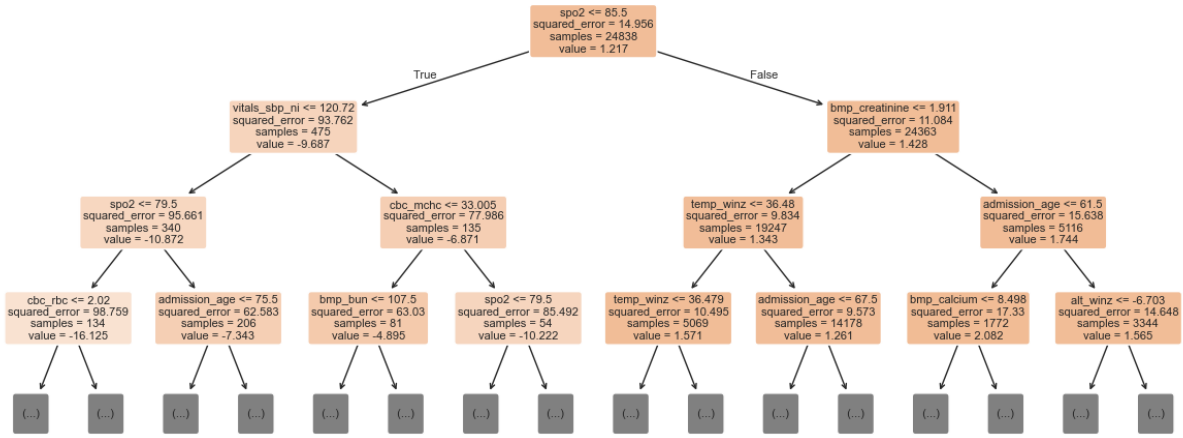
Return to D: Random Forest Regression Analysis , Table of Contents

In [81]:
```
# Initialize and build a Random forest model for the data that was stratifie
rfr_stratified = RandomForestRegressor(
    n_estimators=100,
    max_features="sqrt",  # Square root of the number of features for a regr
    random_state=42,
    oob_score=True
)
rfr_stratified.fit(X_train_imputed_df, y_train)
y_pred_rfr_strat = rfr_stratified.predict(X_test_imputed_df)
```

Visualization of the first tree

In [82]:
```
# First tree of stratified data
# Define feature_names as a list of column names
feature_names_strat = X_train_imputed_df.columns.tolist()

# Extract the first tree from RFR_stratified model
first_tree_strat = rfr_stratified.estimators_[0]
plt.figure(figsize=(15,6))
tree.plot_tree(first_tree_strat,
               feature_names=feature_names_strat,
               fontsize=8,
               filled=True,
               rounded=True,
               max_depth=3)
```

Out[82]:
```
[Text(0.5, 0.9, 'spo2 <= 85.5\nsquared_error = 14.956\nsamples = 24838\nval
ue = 1.217'),
 Text(0.25, 0.7, 'vitals_sbp_ni <= 120.72\nsquared_error = 93.762\nsamples
= 475\nvalue = -9.687'),
 Text(0.375, 0.8, 'True  '),
 Text(0.125, 0.5, 'spo2 <= 79.5\nsquared_error = 95.661\nsamples = 340\nval
ue = -10.872'),
 Text(0.0625, 0.3, 'cbc_rbc <= 2.02\nsquared_error = 98.759\nsamples = 134
\nvalue = -16.125'),
 Text(0.03125, 0.1, '\n  (...)  \n'),
 Text(0.09375, 0.1, '\n  (...)  \n'),
 Text(0.1875, 0.3, 'admission_age <= 75.5\nsquared_error = 62.583\nsamples
= 206\nvalue = -7.343'),
 Text(0.15625, 0.1, '\n  (...)  \n'),
 Text(0.21875, 0.1, '\n  (...)  \n'),
 Text(0.375, 0.5, 'cbc_mchc <= 33.005\nsquared_error = 77.986\nsamples = 13
5\nvalue = -6.871'),
 Text(0.3125, 0.3, 'bmp_bun <= 107.5\nsquared_error = 63.03\nsamples = 81\n
value = -4.895'),
 Text(0.28125, 0.1, '\n  (...)  \n'),
 Text(0.34375, 0.1, '\n  (...)  \n'),
 Text(0.4375, 0.3, 'spo2 <= 79.5\nsquared_error = 85.492\nsamples = 54\nval
ue = -10.222'),
 Text(0.40625, 0.1, '\n  (...)  \n'),
 Text(0.46875, 0.1, '\n  (...)  \n'),
 Text(0.75, 0.7, 'bmp_creatinine <= 1.911\nsquared_error = 11.084\nsamples
= 24363\nvalue = 1.428'),
 Text(0.625, 0.8, '  False'),
 Text(0.625, 0.5, 'temp_winz <= 36.48\nsquared_error = 9.834\nsamples = 192
47\nvalue = 1.343'),
 Text(0.5625, 0.3, 'temp_winz <= 36.479\nsquared_error = 10.495\nsamples =
5069\nvalue = 1.571'),
 Text(0.53125, 0.1, '\n  (...)  \n'),
 Text(0.59375, 0.1, '\n  (...)  \n'),
 Text(0.6875, 0.3, 'admission_age <= 67.5\nsquared_error = 9.573\nsamples =
14178\nvalue = 1.261'),
 Text(0.65625, 0.1, '\n  (...)  \n'),
 Text(0.71875, 0.1, '\n  (...)  \n'),
 Text(0.875, 0.5, 'admission_age <= 61.5\nsquared_error = 15.638\nsamples =
5116\nvalue = 1.744'),
 Text(0.8125, 0.3, 'bmp_calcium <= 8.498\nsquared_error = 17.33\nsamples =
1772\nvalue = 2.082'),
 Text(0.78125, 0.1, '\n  (...)  \n'),
 Text(0.84375, 0.1, '\n  (...)  \n'),
 Text(0.9375, 0.3, 'alt_winz <= -6.703\nsquared_error = 14.648\nsamples = 3
344\nvalue = 1.565'),
 Text(0.90625, 0.1, '\n  (...)  \n'),
 Text(0.96875, 0.1, '\n  (...)  \n')]
```



Evaluation Metrics

Return to D: Analysis , Table of Contents

In [83]:
```
#Evaluation metrics for initial model
mse_strat = round(mean_squared_error(y_test, y_pred_rfr_strat), 2)
rmse_strat = round(np.sqrt(mean_squared_error(y_test, y_pred_rfr_strat)), 2)
# View scores
print("Mean Absolute Error:", round(mean_absolute_error(y_test, y_pred_rfr_s
print("Mean Squared Error:", mse_strat)
print("Root Mean Squared Error:", rmse_strat)
print("R² Score:", round(r2_score(y_test, y_pred_rfr_strat), 4))
```

```
Mean Absolute Error: 2.11
Mean Squared Error: 11.11
Root Mean Squared Error: 3.33
R² Score: 0.2287
```

Feature Selection

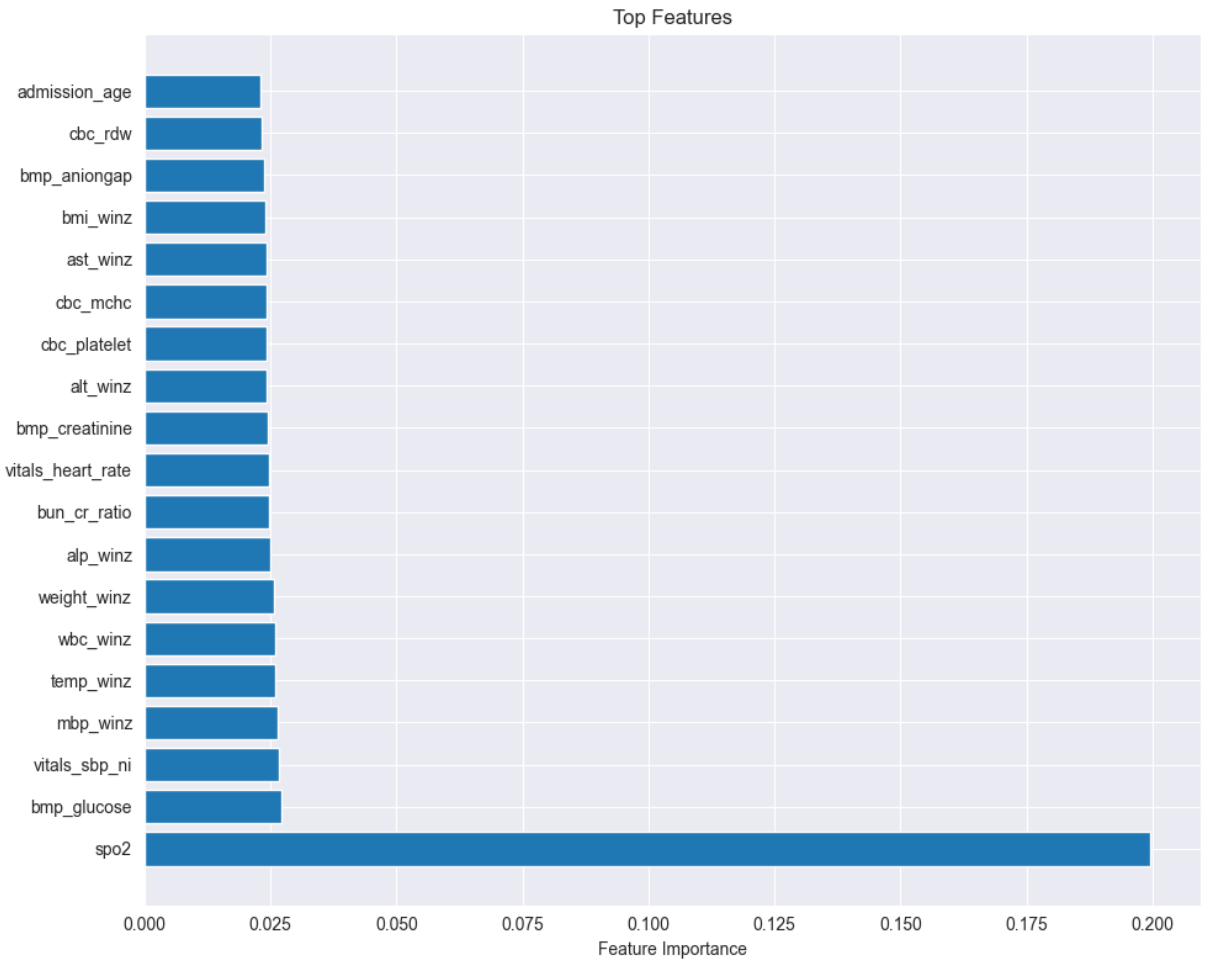
Return to D: Random Forest Regression Analysis , Table of Contents

In [84]:
```
# Select features using only training data to reduce data leakage
selector = SelectFromModel(rfr_stratified, threshold="median")
selector.fit(X_train_imputed_df, y_train)

# Get selected feature names
selected_features = X_train_imputed_df.columns[selector.get_support()]
print(f"Selected {len(selected_features)} features out of {X_train_imputed_d
print(selected_features.tolist())

# Transform data to use only selected features
X_train_selected_strat = selector.transform(X_train_imputed_df)
X_test_selected_strat = selector.transform(X_test_imputed_df)
```

```
Selected 19 features out of 37
['admission_age', 'spo2', 'vitals_heart_rate', 'vitals_sbp_ni', 'cbc_mchc',
'cbc_platelet', 'cbc_rdw', 'bmp_creatinine', 'bmp_glucose', 'bmp_aniongap',
'weight_winz', 'bun_cr_ratio', 'bmi_winz', 'mbp_winz', 'temp_winz', 'wbc_win
z', 'alt_winz', 'alp_winz', 'ast_winz']
```

The model determined that 19 features were important for predictions of the target variable. These features were visualized in order of importance.

In [85]:
```python
# Get feature importance
importances_strat = rfr_stratified.feature_importances_
feature_names_strat = X_train_imputed_df.columns
indices = np.argsort(importances_strat)[::-1]

# Plot only top features
plt.figure(figsize=(10, 8))
plt.barh(range(19), importances_strat[indices][:19])
plt.yticks(range(19), [feature_names_strat[i] for i in indices[:19]])
plt.xlabel('Feature Importance')
plt.title('Top Features')
plt.tight_layout()
plt.show()
```



Optuna Study

An optuna study was done to determine the best hyperparameters to use in the RFR

model.

> **Note**: The output from this code is *tracking* of the process, *not an error*.

Return to D: Random Forest Regression Analysis , Table of Contents

In [86]:
```
# Define the objective function for Optuna
# Time needed - approx 45 min
def objective(trial):
    params = {
        'n_estimators': trial.suggest_int('n_estimators', 100, 800),
        'max_depth': trial.suggest_int('max_depth', 10, 80, log=True),
        'min_samples_split': trial.suggest_int('min_samples_split', 2, 10),
        'min_samples_leaf': trial.suggest_int('min_samples_leaf', 1, 4),
        'max_features': trial.suggest_categorical('max_features', ['sqrt', '
        'max_samples': trial.suggest_categorical('max_samples', [0.7, 0.9, N
    }

    rf = RandomForestRegressor(**params, random_state=42)
    score = cross_val_score(
        rf,
        X_train_selected_strat,
        y_train,
        cv=3,
        scoring='neg_mean_squared_error',
        n_jobs=-1
    )
    # Convert MSE to RMSE
    return np.sqrt(-np.mean(score))  # Square root of MSE = RMSE

study_strat = optuna.create_study(direction='minimize')
study_strat.optimize(objective, n_trials=50)

print("Best trial:")
trial_strat = study_strat.best_trial
print("  Value: {}".format(trial_strat.value))
print("  Params: ")
for key, value in trial_strat.params.items():
    print("    {}: {}".format(key, value))
```

```
[I 2025-03-31 08:39:48,895] A new study created in memory with name: no-nam
e-f590f5fb-c105-4473-a16d-95b33fb7923d
[I 2025-03-31 08:40:05,892] Trial 0 finished with value: 3.4277416450202116
and parameters: {'n_estimators': 233, 'max_depth': 27, 'min_samples_split':
4, 'min_samples_leaf': 2, 'max_features': 'sqrt', 'max_samples': None}. Best
is trial 0 with value: 3.4277416450202116.
[I 2025-03-31 08:40:34,996] Trial 1 finished with value: 3.4189183216435963
and parameters: {'n_estimators': 464, 'max_depth': 10, 'min_samples_split':
5, 'min_samples_leaf': 1, 'max_features': 0.5, 'max_samples': 0.9}. Best is
trial 1 with value: 3.4189183216435963.
[I 2025-03-31 08:41:19,952] Trial 2 finished with value: 3.4146876471638263
and parameters: {'n_estimators': 622, 'max_depth': 11, 'min_samples_split':
4, 'min_samples_leaf': 4, 'max_features': 0.5, 'max_samples': None}. Best is
trial 2 with value: 3.4146876471638263.
[I 2025-03-31 08:41:36,073] Trial 3 finished with value: 3.4267664922279706
and parameters: {'n_estimators': 238, 'max_depth': 62, 'min_samples_split':
7, 'min_samples_leaf': 2, 'max_features': 'log2', 'max_samples': 0.9}. Best
is trial 2 with value: 3.4146876471638263.
[I 2025-03-31 08:42:04,468] Trial 4 finished with value: 3.4249425734069714
and parameters: {'n_estimators': 396, 'max_depth': 42, 'min_samples_split':
9, 'min_samples_leaf': 2, 'max_features': 'sqrt', 'max_samples': None}. Best
is trial 2 with value: 3.4146876471638263.
[I 2025-03-31 08:42:41,762] Trial 5 finished with value: 3.423620796021168 a
nd parameters: {'n_estimators': 550, 'max_depth': 60, 'min_samples_split': 8
, 'min_samples_leaf': 2, 'max_features': 'log2', 'max_samples': 0.9}. Best i
s trial 2 with value: 3.4146876471638263.
[I 2025-03-31 08:43:35,645] Trial 6 finished with value: 3.418873368725711 a
nd parameters: {'n_estimators': 480, 'max_depth': 18, 'min_samples_split': 7
, 'min_samples_leaf': 3, 'max_features': 0.5, 'max_samples': None}. Best is
trial 2 with value: 3.4146876471638263.
[I 2025-03-31 08:45:16,807] Trial 7 finished with value: 3.421594264390928 a
nd parameters: {'n_estimators': 725, 'max_depth': 27, 'min_samples_split': 8
, 'min_samples_leaf': 3, 'max_features': 0.5, 'max_samples': None}. Best is
trial 2 with value: 3.4146876471638263.
[I 2025-03-31 08:45:43,066] Trial 8 finished with value: 3.433318994322814 a
nd parameters: {'n_estimators': 175, 'max_depth': 68, 'min_samples_split': 1
0, 'min_samples_leaf': 2, 'max_features': 0.5, 'max_samples': None}. Best is
trial 2 with value: 3.4146876471638263.
[I 2025-03-31 08:46:10,343] Trial 9 finished with value: 3.4282595325561096
and parameters: {'n_estimators': 698, 'max_depth': 15, 'min_samples_split':
3, 'min_samples_leaf': 2, 'max_features': 'sqrt', 'max_samples': 0.7}. Best
is trial 2 with value: 3.4146876471638263.
[I 2025-03-31 08:47:27,895] Trial 10 finished with value: 3.417805660127464
and parameters: {'n_estimators': 618, 'max_depth': 11, 'min_samples_split':
2, 'min_samples_leaf': 4, 'max_features': None, 'max_samples': 0.7}. Best is
trial 2 with value: 3.4146876471638263.
[I 2025-03-31 08:48:38,407] Trial 11 finished with value: 3.4171930618228967
and parameters: {'n_estimators': 609, 'max_depth': 10, 'min_samples_split':
2, 'min_samples_leaf': 4, 'max_features': None, 'max_samples': 0.7}. Best is
trial 2 with value: 3.4146876471638263.
[I 2025-03-31 08:50:57,770] Trial 12 finished with value: 3.4214718777800557
and parameters: {'n_estimators': 796, 'max_depth': 16, 'min_samples_split':
2, 'min_samples_leaf': 4, 'max_features': None, 'max_samples': 0.7}. Best is
trial 2 with value: 3.4146876471638263.
[I 2025-03-31 08:52:25,812] Trial 13 finished with value: 3.4196419421087567
and parameters: {'n_estimators': 604, 'max_depth': 13, 'min_samples_split':
```

4, 'min_samples_leaf': 4, 'max_features': None, 'max_samples': 0.7}. Best is trial 2 with value: 3.4146876471638263.
[I 2025-03-31 08:53:41,894] Trial 14 finished with value: 3.4293601163882577 and parameters: {'n_estimators': 357, 'max_depth': 21, 'min_samples_split': 5, 'min_samples_leaf': 3, 'max_features': None, 'max_samples': 0.7}. Best is trial 2 with value: 3.4146876471638263.
[I 2025-03-31 08:54:21,033] Trial 15 finished with value: 3.4162930650526495 and parameters: {'n_estimators': 555, 'max_depth': 10, 'min_samples_split': 3, 'min_samples_leaf': 4, 'max_features': 0.5, 'max_samples': None}. Best is trial 2 with value: 3.4146876471638263.
[I 2025-03-31 08:56:09,111] Trial 16 finished with value: 3.419450323094758 and parameters: {'n_estimators': 544, 'max_depth': 38, 'min_samples_split': 4, 'min_samples_leaf': 4, 'max_features': 0.5, 'max_samples': None}. Best is trial 2 with value: 3.4146876471638263.
[I 2025-03-31 08:56:39,848] Trial 17 finished with value: 3.41746868457422 and parameters: {'n_estimators': 339, 'max_depth': 13, 'min_samples_split': 5, 'min_samples_leaf': 3, 'max_features': 0.5, 'max_samples': None}. Best is trial 2 with value: 3.4146876471638263.
[I 2025-03-31 08:59:32,659] Trial 18 finished with value: 3.433028152372272 and parameters: {'n_estimators': 702, 'max_depth': 20, 'min_samples_split': 3, 'min_samples_leaf': 1, 'max_features': 0.5, 'max_samples': None}. Best is trial 2 with value: 3.4146876471638263.
[I 2025-03-31 09:00:24,970] Trial 19 finished with value: 3.4153259370773665 and parameters: {'n_estimators': 518, 'max_depth': 13, 'min_samples_split': 6, 'min_samples_leaf': 4, 'max_features': 0.5, 'max_samples': None}. Best is trial 2 with value: 3.4146876471638263.
[I 2025-03-31 09:00:41,624] Trial 20 finished with value: 3.4312020304475905 and parameters: {'n_estimators': 414, 'max_depth': 13, 'min_samples_split': 6, 'min_samples_leaf': 3, 'max_features': 'log2', 'max_samples': None}. Best is trial 2 with value: 3.4146876471638263.
[I 2025-03-31 09:01:23,819] Trial 21 finished with value: 3.4150717214286437 and parameters: {'n_estimators': 522, 'max_depth': 12, 'min_samples_split': 6, 'min_samples_leaf': 4, 'max_features': 0.5, 'max_samples': None}. Best is trial 2 with value: 3.4146876471638263.
[I 2025-03-31 09:02:07,215] Trial 22 finished with value: 3.415276117987006 and parameters: {'n_estimators': 533, 'max_depth': 12, 'min_samples_split': 6, 'min_samples_leaf': 4, 'max_features': 0.5, 'max_samples': None}. Best is trial 2 with value: 3.4146876471638263.
[I 2025-03-31 09:03:30,299] Trial 23 finished with value: 3.417861782374553 and parameters: {'n_estimators': 646, 'max_depth': 22, 'min_samples_split': 7, 'min_samples_leaf': 4, 'max_features': 0.5, 'max_samples': None}. Best is trial 2 with value: 3.4146876471638263.
[I 2025-03-31 09:04:25,157] Trial 24 finished with value: 3.4178529188471587 and parameters: {'n_estimators': 510, 'max_depth': 16, 'min_samples_split': 6, 'min_samples_leaf': 3, 'max_features': 0.5, 'max_samples': None}. Best is trial 2 with value: 3.4146876471638263.
[I 2025-03-31 09:05:26,565] Trial 25 finished with value: 3.4144803305538245 and parameters: {'n_estimators': 793, 'max_depth': 12, 'min_samples_split': 5, 'min_samples_leaf': 4, 'max_features': 0.5, 'max_samples': 0.9}. Best is trial 25 with value: 3.4144803305538245.
[I 2025-03-31 09:06:40,397] Trial 26 finished with value: 3.41618799572093 and parameters: {'n_estimators': 775, 'max_depth': 15, 'min_samples_split': 5, 'min_samples_leaf': 4, 'max_features': 0.5, 'max_samples': 0.9}. Best is trial 25 with value: 3.4144803305538245.
[I 2025-03-31 09:08:03,506] Trial 27 finished with value: 3.4173752728342537 and parameters: {'n_estimators': 739, 'max_depth': 18, 'min_samples_split':

4, 'min_samples_leaf': 3, 'max_features': 0.5, 'max_samples': 0.9}. Best is trial 25 with value: 3.4144803305538245.
[I 2025-03-31 09:08:44,278] Trial 28 finished with value: 3.4204249806727938 and parameters: {'n_estimators': 666, 'max_depth': 24, 'min_samples_split': 5, 'min_samples_leaf': 4, 'max_features': 'sqrt', 'max_samples': 0.9}. Best is trial 25 with value: 3.4144803305538245.
[I 2025-03-31 09:09:35,452] Trial 29 finished with value: 3.4232682068240896 and parameters: {'n_estimators': 753, 'max_depth': 30, 'min_samples_split': 4, 'min_samples_leaf': 3, 'max_features': 'log2', 'max_samples': 0.9}. Best is trial 25 with value: 3.4144803305538245.
[I 2025-03-31 09:09:39,361] Trial 30 finished with value: 3.4492500561473367 and parameters: {'n_estimators': 100, 'max_depth': 11, 'min_samples_split': 3, 'min_samples_leaf': 4, 'max_features': 'sqrt', 'max_samples': 0.9}. Best is trial 25 with value: 3.4144803305538245.
[I 2025-03-31 09:10:27,912] Trial 31 finished with value: 3.415291184740945 and parameters: {'n_estimators': 573, 'max_depth': 12, 'min_samples_split': 6, 'min_samples_leaf': 4, 'max_features': 0.5, 'max_samples': None}. Best is trial 25 with value: 3.4144803305538245.
[I 2025-03-31 09:11:01,393] Trial 32 finished with value: 3.4150073385795374 and parameters: {'n_estimators': 438, 'max_depth': 11, 'min_samples_split': 5, 'min_samples_leaf': 4, 'max_features': 0.5, 'max_samples': None}. Best is trial 25 with value: 3.4144803305538245.
[I 2025-03-31 09:11:38,311] Trial 33 finished with value: 3.414703228547349 and parameters: {'n_estimators': 460, 'max_depth': 11, 'min_samples_split': 5, 'min_samples_leaf': 4, 'max_features': 0.5, 'max_samples': None}. Best is trial 25 with value: 3.4144803305538245.
[I 2025-03-31 09:11:58,827] Trial 34 finished with value: 3.417329551300044 and parameters: {'n_estimators': 311, 'max_depth': 10, 'min_samples_split': 5, 'min_samples_leaf': 4, 'max_features': 0.5, 'max_samples': 0.9}. Best is trial 25 with value: 3.4144803305538245.
[I 2025-03-31 09:12:42,540] Trial 35 finished with value: 3.4232344166903683 and parameters: {'n_estimators': 452, 'max_depth': 14, 'min_samples_split': 5, 'min_samples_leaf': 1, 'max_features': 0.5, 'max_samples': None}. Best is trial 25 with value: 3.4144803305538245.
[I 2025-03-31 09:13:03,806] Trial 36 finished with value: 3.424586128313503 and parameters: {'n_estimators': 408, 'max_depth': 18, 'min_samples_split': 4, 'min_samples_leaf': 3, 'max_features': 'log2', 'max_samples': 0.9}. Best is trial 25 with value: 3.4144803305538245.
[I 2025-03-31 09:13:47,780] Trial 37 finished with value: 3.4213066565581314 and parameters: {'n_estimators': 305, 'max_depth': 49, 'min_samples_split': 4, 'min_samples_leaf': 4, 'max_features': 0.5, 'max_samples': None}. Best is trial 25 with value: 3.4144803305538245.
[I 2025-03-31 09:14:23,324] Trial 38 finished with value: 3.4146371672689853 and parameters: {'n_estimators': 464, 'max_depth': 11, 'min_samples_split': 7, 'min_samples_leaf': 4, 'max_features': 0.5, 'max_samples': None}. Best is trial 25 with value: 3.4144803305538245.
[I 2025-03-31 09:14:55,568] Trial 39 finished with value: 3.421770061171141 and parameters: {'n_estimators': 478, 'max_depth': 34, 'min_samples_split': 7, 'min_samples_leaf': 3, 'max_features': 'sqrt', 'max_samples': 0.9}. Best is trial 25 with value: 3.4144803305538245.
[I 2025-03-31 09:15:34,644] Trial 40 finished with value: 3.421620286946302 and parameters: {'n_estimators': 268, 'max_depth': 79, 'min_samples_split': 8, 'min_samples_leaf': 4, 'max_features': 0.5, 'max_samples': None}. Best is trial 25 with value: 3.4144803305538245.
[I 2025-03-31 09:16:04,298] Trial 41 finished with value: 3.416879259779294 and parameters: {'n_estimators': 425, 'max_depth': 10, 'min_samples_split':

```
7, 'min_samples_leaf': 4, 'max_features': 0.5, 'max_samples': None}. Best is
trial 25 with value: 3.4144803305538245.
[I 2025-03-31 09:16:35,736] Trial 42 finished with value: 3.4162990616994238
and parameters: {'n_estimators': 374, 'max_depth': 12, 'min_samples_split':
5, 'min_samples_leaf': 4, 'max_features': 0.5, 'max_samples': None}. Best is
trial 25 with value: 3.4144803305538245.
[I 2025-03-31 09:17:12,734] Trial 43 finished with value: 3.414688620525028
and parameters: {'n_estimators': 480, 'max_depth': 11, 'min_samples_split':
8, 'min_samples_leaf': 4, 'max_features': 0.5, 'max_samples': None}. Best is
trial 25 with value: 3.4144803305538245.
[I 2025-03-31 09:17:50,928] Trial 44 finished with value: 3.4155207801751217
and parameters: {'n_estimators': 486, 'max_depth': 11, 'min_samples_split':
9, 'min_samples_leaf': 4, 'max_features': 0.5, 'max_samples': None}. Best is
trial 25 with value: 3.4144803305538245.
[I 2025-03-31 09:18:58,425] Trial 45 finished with value: 3.414745460277686
and parameters: {'n_estimators': 656, 'max_depth': 15, 'min_samples_split':
8, 'min_samples_leaf': 4, 'max_features': 0.5, 'max_samples': None}. Best is
trial 25 with value: 3.4144803305538245.
[I 2025-03-31 09:19:12,147] Trial 46 finished with value: 3.42982696898907 a
nd parameters: {'n_estimators': 379, 'max_depth': 14, 'min_samples_split': 1
0, 'min_samples_leaf': 3, 'max_features': 'log2', 'max_samples': 0.7}. Best
is trial 25 with value: 3.4144803305538245.
[I 2025-03-31 09:21:27,631] Trial 47 finished with value: 3.437175836988157
and parameters: {'n_estimators': 572, 'max_depth': 17, 'min_samples_split':
9, 'min_samples_leaf': 2, 'max_features': None, 'max_samples': None}. Best i
s trial 25 with value: 3.4144803305538245.
[I 2025-03-31 09:22:03,330] Trial 48 finished with value: 3.4147275582942 an
d parameters: {'n_estimators': 487, 'max_depth': 11, 'min_samples_split': 8,
'min_samples_leaf': 4, 'max_features': 0.5, 'max_samples': 0.9}. Best is tri
al 25 with value: 3.4144803305538245.
[I 2025-03-31 09:22:30,383] Trial 49 finished with value: 3.4285146851756165
and parameters: {'n_estimators': 592, 'max_depth': 14, 'min_samples_split':
7, 'min_samples_leaf': 4, 'max_features': 'sqrt', 'max_samples': None}. Best
is trial 25 with value: 3.4144803305538245.
Best trial:
  Value: 3.4144803305538245
  Params:
    n_estimators: 793
    max_depth: 12
    min_samples_split: 5
    min_samples_leaf: 4
    max_features: 0.5
    max_samples: 0.9
```

Due to the amount of time that was taken to run the optimization search, the study was saved to avoid re-running if the notebook had to be re-run at any point in the analysis.

In [88]:
```python
# Save the completed study
joblib.dump(study_strat, 'optuna_strat.pkl')

# To load it later:
#loaded_study = joblib.load('optuna_strat.pkl')
```

Out[88]:
```
['optuna_strat.pkl']
```

Final Model

The final model was fit and trained on the data using the best fit determined from the Optuna study. The data used for this model was the top 19 features selected from the initial model.

Return to D: Random Forest Regression Analysis , Table of Contents

In [91]:
```
# Train the best model on the selected features training set
#best_params_strat = loaded_study.best_params # If using loaded study
best_params_strat = study_strat.best_params
best_rf_strat = RandomForestRegressor(**best_params_strat, random_state=42)
best_rf_strat.fit(X_train_selected_strat, y_train)
```

Out[91]:
```
RandomForestRegressor(max_depth=12, max_features=0.5, max_samples=
0.9,
                      min_samples_leaf=4, min_samples_split=5, n_es
timators=793,
                      random_state=42)
```

Make Predictions

The test data was then fit to the final model and evaluated using the same four metrics as the initial model.

Return to D: Random Forest Regression Analysis , Table of Contents

In [92]:
```
# Make predictions using the test data
y_pred_optuna_strat = best_rf_strat.predict(X_test_selected_strat)
```

In [93]:
```
#Evaluation metrics
mse_optuna_strat = round(mean_squared_error(y_test, y_pred_optuna_strat), 2)
rmse_optuna_strat = round(np.sqrt(mean_squared_error(y_test, y_pred_optuna_s
# View accuracy scores
print("Mean Absolute Error:", round(mean_absolute_error(y_test, y_pred_optun
print("Mean Squared Error:", mse_optuna_strat)
print("Root Mean Squared Error:", rmse_optuna_strat)
print("R² Score:", round(r2_score(y_test, y_pred_optuna_strat), 4))
```

```
Mean Absolute Error: 2.08
Mean Squared Error: 10.83
Root Mean Squared Error: 3.29
R² Score: 0.2483
```

Model Comparison

Comparison of the basic model to the final model

Metric	Model 1	Model 2
Mean Absolute Error (MAE)	2.13	2.08
Mean Squared Error (MSE)	11.12	10.83
Root Mean Squared Error (RMSE)	3.33	3.29
R^2 Score	0.2279	0.2483

Return to D: Random Forest Regression Analysis , Table of Contents

E: Data Summary and Implications

The model was unable to predict SpO2 - SaO2 gap value, as shown by the RMSE. While it was close with a final value of 3.29, this is slightly worse than the current FDA standard for pulse oximetry accuracy requirements. The null hypothesis of a random forest model cannot detect the difference between the SpO2 and the SaO2 with an RMSE of less than 3 percentage points failed to be rejected.

The final model did show improvement over the initial model, with the actual improvement being only a difference of 0.04 RMSE.

For the final model, the MAE was 2.08, meaning that on average, the error was 2%. In the medical realm, that amount can be the difference between the diagnosis of hypoxemia and all the treatments that accompany the diagnosis. There was also an improvement in the MSE, although it remained high. The R2 score also remained very low. Overall, these metrics revealed that the RFR model was not able to predict the actual SpO2 values appropriately.

A confusion matrix was also done to evaluate the accuracy and recall of the model. This RFR can be considered a “model” study in why accuracy should not be the only metric used for evaluation. This accuracy of the final RFR was 84%, which on the surface seems like a robust value. But when broken down, it is shown how poorly this model actually performed.

The class of predictions that were within the FDA error tolerance had values within three SpO2 percentage points and will be referred to as “within tolerance.” The values that were equal or greater than 3 percentage points can be the group at risk for hidden hypoxemia and will be referred to as the “at risk” group.

Confusion Matrix Results:

- 8205 correctly predicted "within Tolerance" | 67 incorrectly predicted "at Risk"
- 1525 incorrectly predicted "within Tolerance" | 22 correctly predicted "at Risk"

The number of true negatives was 8205, meaning that “within tolerance” was correctly

predicted. However, there were 1525 false negatives, meaning the model predicted "within tolerance" when the result was actually "at risk."

There were 67 false positives, meaning that “at risk” was predicted incorrectly for “within tolerance” cases. The number of true positives was only 22, meaning the model only correctly predicted these cases as “at risk.”

The model was great at predicting “within tolerance” values and was able to do so 99% of the time (the recall metric) but only identified "at risk" cases 1% of the time. This striking imbalance reveals that despite “good” overall accuracy of 84%, the model fails to identify patients who might have hidden hypoxemia.

With the analysis result not offering an improvement in SpO2 analysis, the recommendation is to continue with the current process of using SaO2 values as the standard for determining hypoxemia. While the current process has shortcomings, creating any changes based on this model will add to the number of missed hypoxemia cases. Within the understanding of the deficits of SpO2 monitoring, it may be worthwhile to create educational opportunities to re-enforce the concept of using all available resources and critical thinking to make clinical decisions rather than relying on a single tool.

A limitation of this analysis was the data was over a 24-hour period, which may have actually rendered the usefulness of the laboratory tests to be near null. Within an ICU setting, laboratory tests are often drawn within four hours of interventions. A hematocrit drawn eight hours after the SpO2 reading was recorded will have little real-life value. SpO2 is a continuous measurement with many changing variables, yet this analysis can be considered static, again drawing away from usefulness in a dynamic clinical setting.

In the future, improvements in this model can take on a couple of directions. The first would be to incorporate venous blood gas (VBG) values into the analysis. This is a non-arterial blood draw that can add a large amount of information to an RFR model, while remaining relatively low risk for the patient since it can be drawn at the same time an IV is started. While VBGs are not a replacement for ABG and SaO2, it may be able to add enough data to allow the model to make accurate predictions. It may also be helpful to use data that incorporates the use of medications, types of oxygen intervention, as well as the amounts of oxygen and the medication dosages. Vasoactive drugs are often administered to critically ill patients and can decrease peripheral blood flow, affecting SpO2 readings. Supplemental oxygen and ventilators also have an effect on the SpO2 and SaO2 measurements. This would add to the robustness of the data, allowing for a better overall understanding of the patients' condition.

A second direction that could be taken with this data would be to shift away from regression and tree models since this RFR model did not have strong results. A neural network used to predict the hidden hypoxia question which would be framed as a classification problem. By simplifying the prediction to binary results, it would also

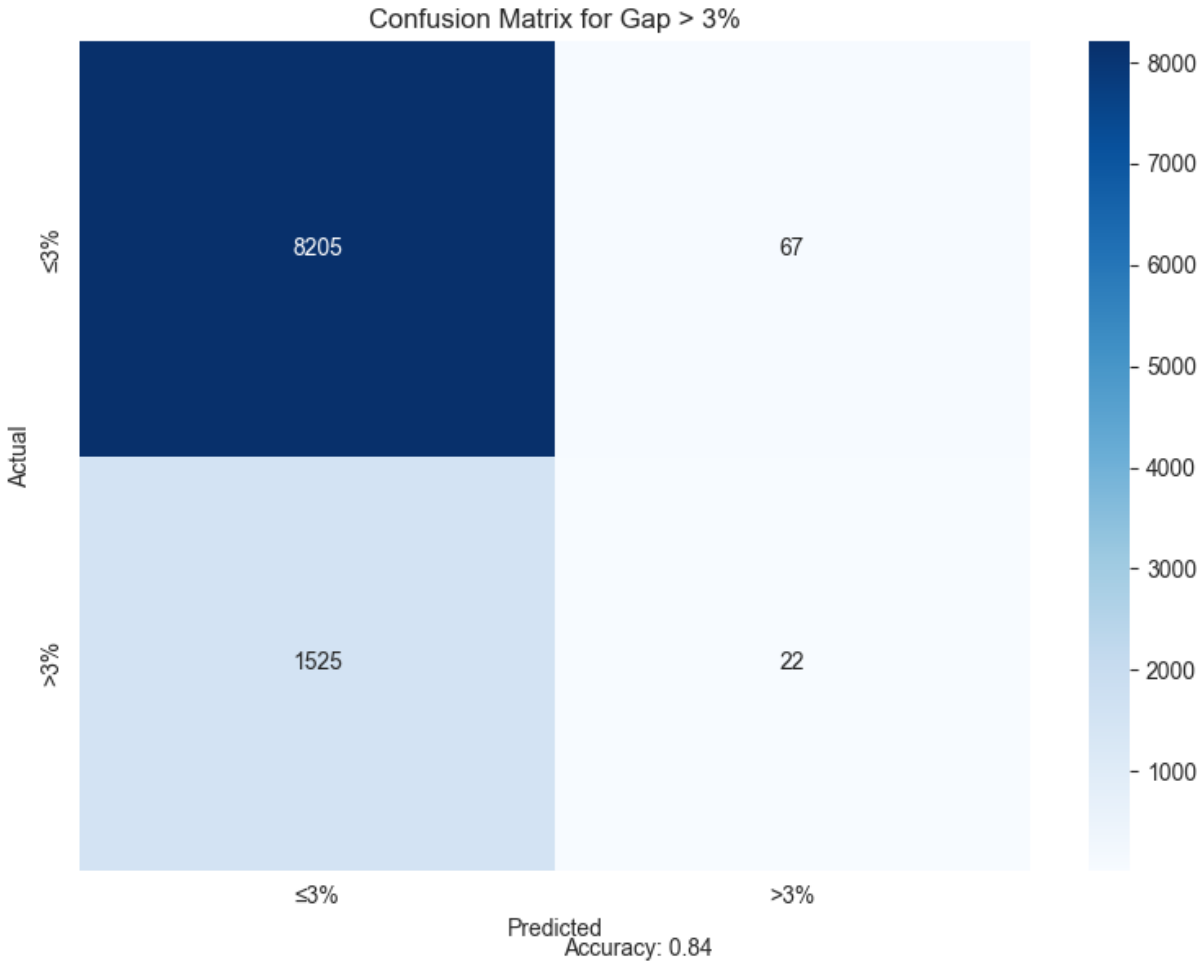
simplify it for the bedside clinician who would not have to interpret the results based on a numeric outcome. Neural networks do well with large data sets as well as finding complex non-linear relationships within data.

Confusion Matrix

In [94]:
```
# Convert gap to binary outcome
y_true_binary = (y_test > 3).astype(int)  # 1 if gap > 3%, 0 otherwise

# Convert predictions to binary
y_pred_binary = (y_pred_optuna_strat > 3).astype(int)

# Calculate metrics
from sklearn.metrics import classification_report, confusion_matrix
print(confusion_matrix(y_true_binary, y_pred_binary))
print(classification_report(y_true_binary, y_pred_binary))
```

```
[[8205   67]
 [1525   22]]
              precision    recall  f1-score   support

           0       0.84      0.99      0.91      8272
           1       0.25      0.01      0.03      1547

    accuracy                           0.84      9819
   macro avg       0.55      0.50      0.47      9819
weighted avg       0.75      0.84      0.77      9819
```

In [95]:
```
# Calculate confusion matrix
cm = confusion_matrix(y_true_binary, y_pred_binary)
# Create heatmap
plt.figure(figsize=(8, 6))
sns.heatmap(cm, annot=True, fmt='d', cmap='Blues', cbar=True)

# Add labels
plt.xlabel('Predicted')
plt.ylabel('Actual')
plt.title('Confusion Matrix for Gap > 3%')
plt.xticks([0.5, 1.5], ['≤3%', '>3%'])
plt.yticks([0.5, 1.5], ['≤3%', '>3%'])
# Add accuracy as text
accuracy = np.trace(cm) / np.sum(cm)
plt.figtext(0.5, 0.01, f'Accuracy: {accuracy:.2f}', ha='center')

plt.tight_layout()
plt.show()

# Print the classification report
print(classification_report(y_true_binary, y_pred_binary))
```



	precision	recall	f1-score	support
0	0.84	0.99	0.91	8272
1	0.25	0.01	0.03	1547
accuracy			0.84	9819
macro avg	0.55	0.50	0.47	9819
weighted avg	0.75	0.84	0.77	9819

Visualizations

Although not required for the analysis, several visualizations were done using the SHAP package to better understand the relationships between the target variable and the included features. SHAP can help simplify machine learning outputs in an intuitive and accessible manner. These visualizations were created with the intention of using some in the presentation of findings.

Return to D: Random Forest Regression Analysis , Table of Contents

In [96]:
```
#Use SHAP (SHapley Additive exPlanations)
explainer = shap.TreeExplainer(best_rf_strat)
```

In [97]:
```
shap_values = explainer.shap_values(X_test_selected_strat)
```

```
X_test_shap_df = pd.DataFrame(X_test_selected_strat, columns= selected_featu
```

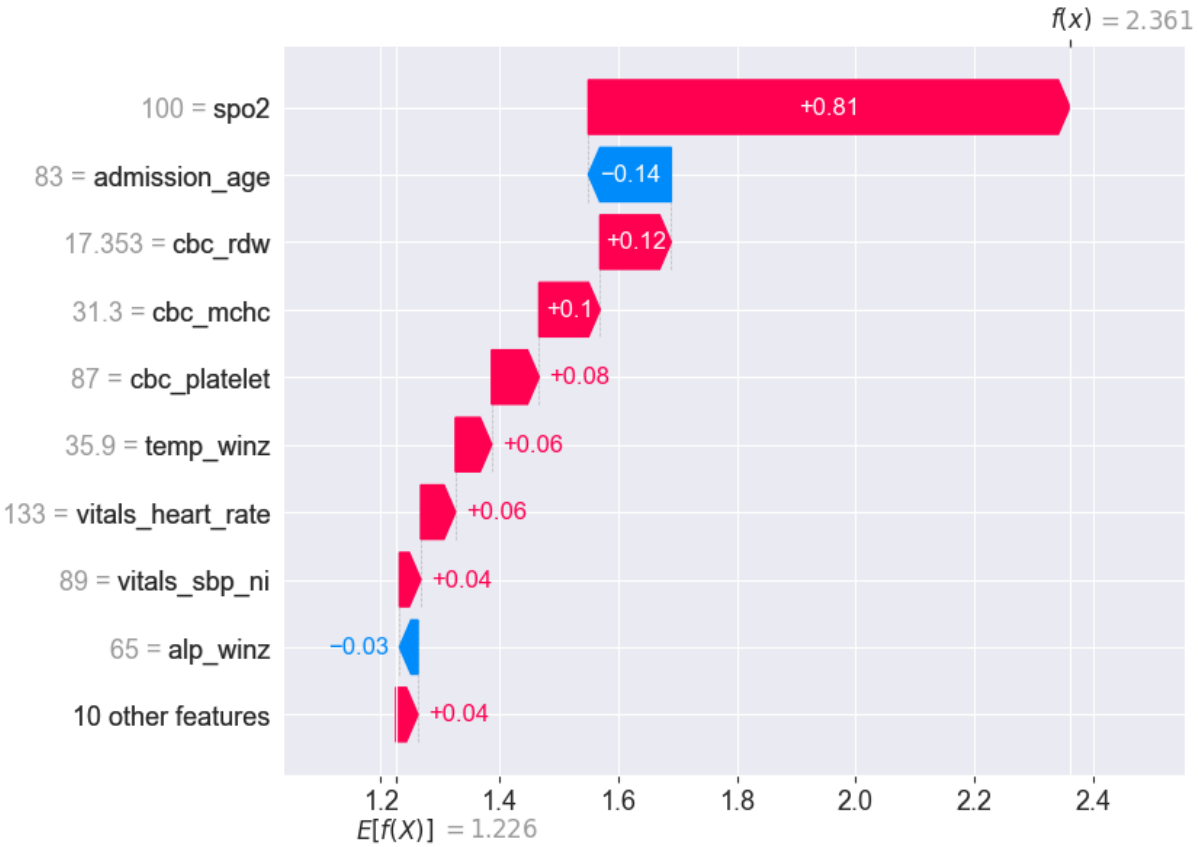
In [98]:
```
#Interaction values - this takes a while ( ~ 3 hr 40 min)
shap_interaction_values = explainer.shap_interaction_values(X_test_shap_df)
```

In [ ]:
```
shap_explanation = explainer(X_test_shap_df)
```

Waterfall plot for a single observation

This plot illustrates feature influence for a single row. The predicted gap is 2.36. The X axis shows the average value of the gap, which in this instance is 1.2. SpO2 has the greatest influence on this prediction, with age having the second-highest influence, although it moved the gap value in a negative direction. This is an example of how the features are working within the model for each observation.

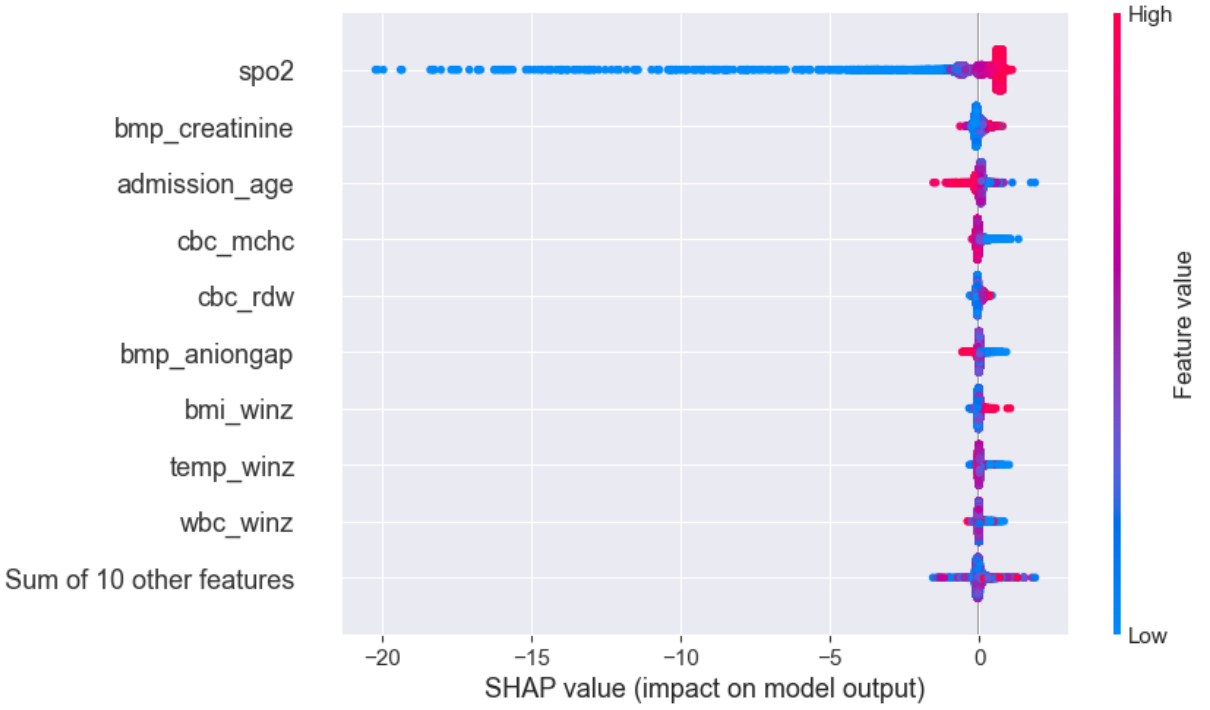
In [100…
```
#Single row
import warnings
warnings.filterwarnings("ignore", category=FutureWarning)
exp = shap.Explanation(values=shap_values,
                       base_values=explainer.expected_value,
                       data=X_test_shap_df.values,
                       feature_names=X_test_shap_df.columns)

# waterfall plot
shap.plots.waterfall(exp[0])
```



Beeswarm plot

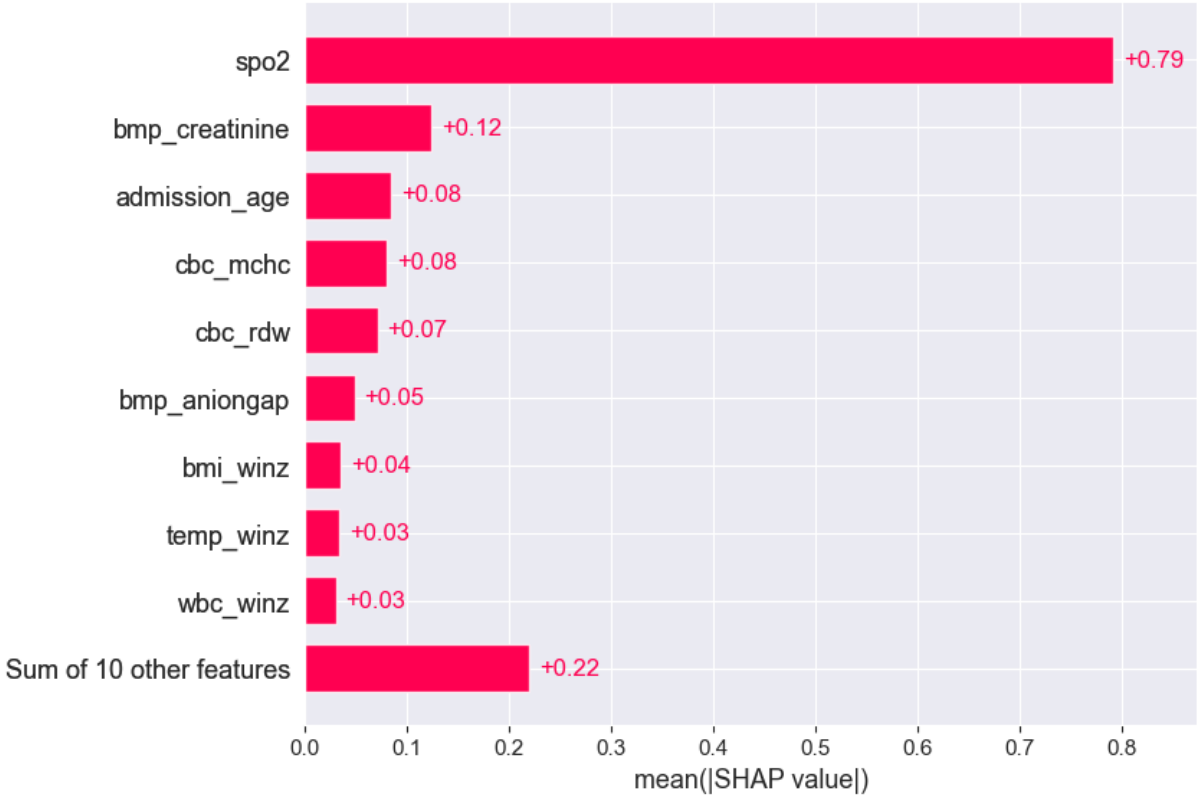
It can become tedious to view each observation, so a beeswarm plot allows for the interpretation of feature influence on the model as a whole. Each dot represents a specific instance from the data set. The SHAP values are shown on the X axis and explain how much each feature moves the prediction away from the baseline, with negative values decreasing the prediction, and positive values moving the prediction in a positive direction. The plot below illustrates that lower SPO2 values, which are shown in blue have a stronger negative influence on the prediction, while lower admission age has a positive influence. This type of visualization is helpful to identify the patterns of influence of the variables.

In [101…
```
# Plot of the features
shap.plots.beeswarm(shap_explanation)
```



The bar plot below shows the strength of the influence a feature had on the prediction, regardless of the direction of that influence.

In [102…
```
shap.plots.bar(shap_explanation)
```

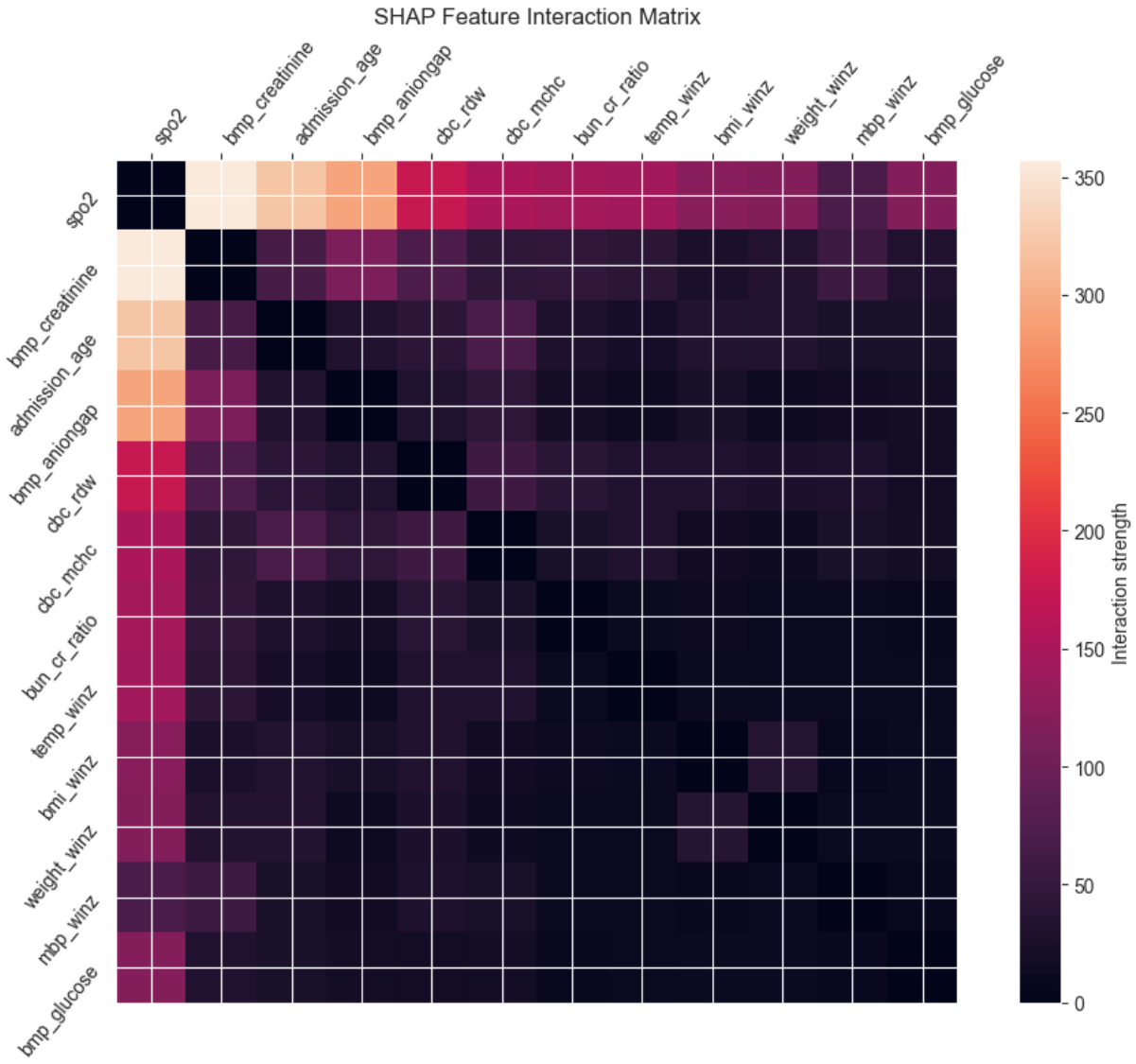


An interaction matrix was created to look at more than just the relationship the features had with the target variable, but with one another. The Matrix revealed a strong relationship between SpO2 and creatinine, age and the anion gap.

In [103…

```
# Create interaction matrix
interaction_matrix = np.abs(shap_interaction_values).sum(0)

for i in range(interaction_matrix.shape[0]):
    interaction_matrix[i, i] = 0

# Get indices of top features by interaction strength
inds = np.argsort(-interaction_matrix.sum(0))[:12]
sorted_ia_matrix = interaction_matrix[inds, :][:, inds]

plt.figure(figsize=(10, 8))
plt.imshow(sorted_ia_matrix)
plt.yticks(
    range(sorted_ia_matrix.shape[0]),
    X_test_shap_df.columns[inds],
    rotation=50.4,
    horizontalalignment="right",
)
plt.xticks(
    range(sorted_ia_matrix.shape[0]),
    X_test_shap_df.columns[inds],
    rotation=50.4,
    horizontalalignment="left",
)
plt.gca().xaxis.tick_top()
plt.colorbar(label='Interaction strength')
```

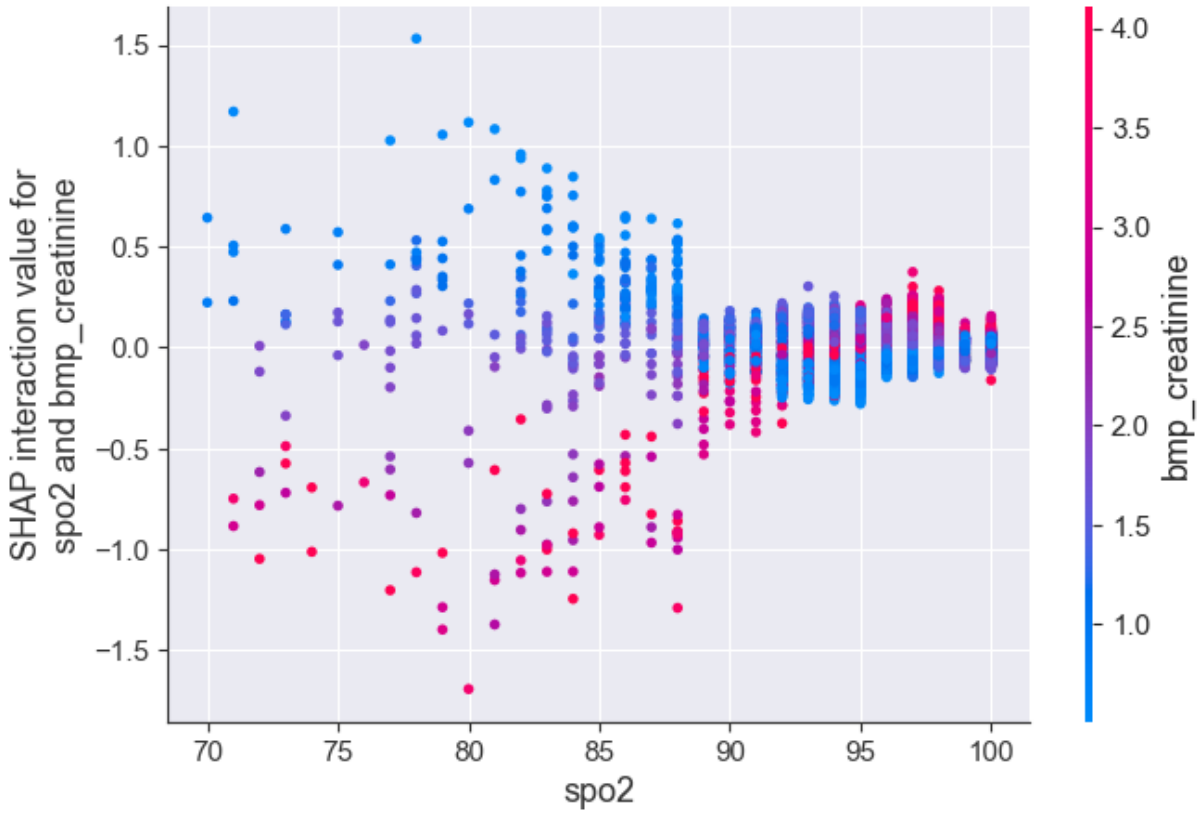
```
plt.title('SHAP Feature Interaction Matrix')
plt.tight_layout()
plt.show()
```



The next two plots show the interaction between SpO2 and creatinine and SpO2 and age. For both of these plots, it can be noted that the variability in the amount of SHAP influence increases greatly around the 89–90% SpO2 level. This aligns clinically with lower SpO2 values having a greater chance of being incorrect when using pulse oximetry.

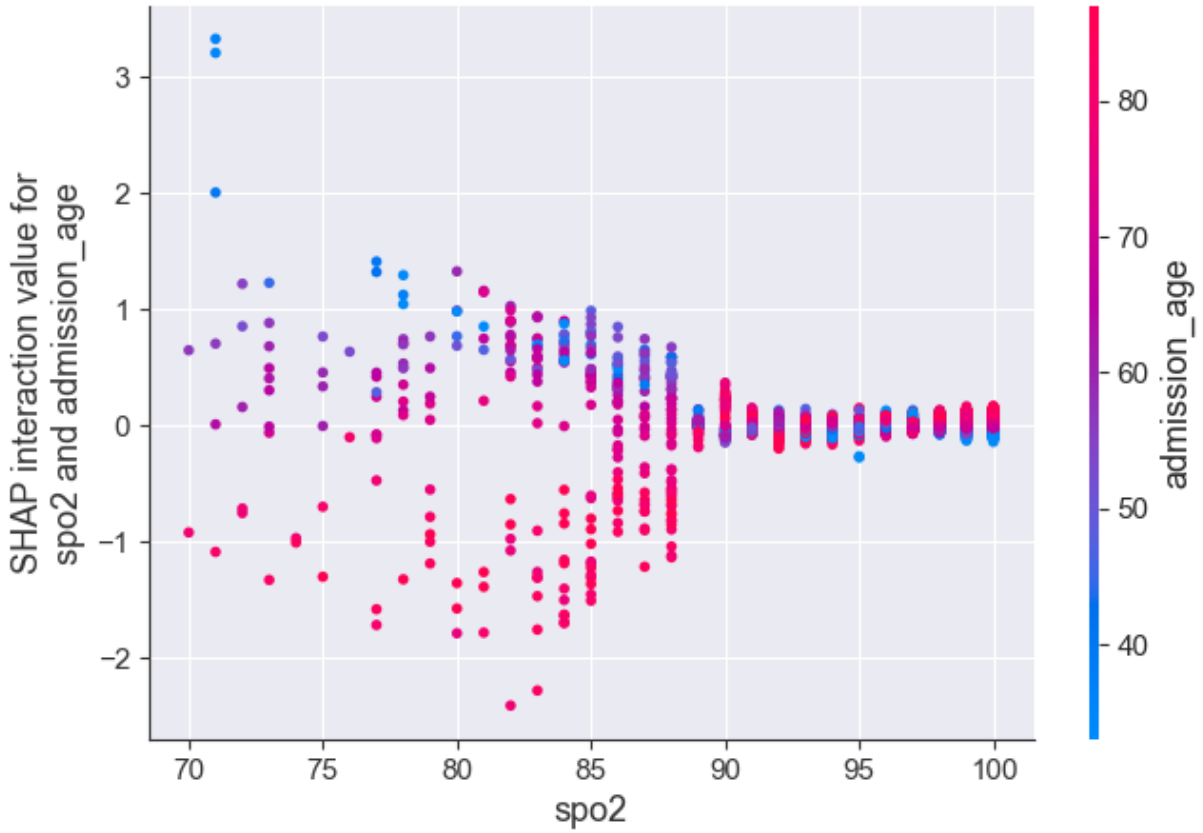
In [104…

```
#SpO2 and Creatinine interaction
spo2_idx = list(X_test_shap_df.columns).index('spo2')
creatinine_idx = list(X_test_shap_df.columns).index('bmp_creatinine')
shap.dependence_plot((spo2_idx, creatinine_idx), shap_interaction_values, X_
```



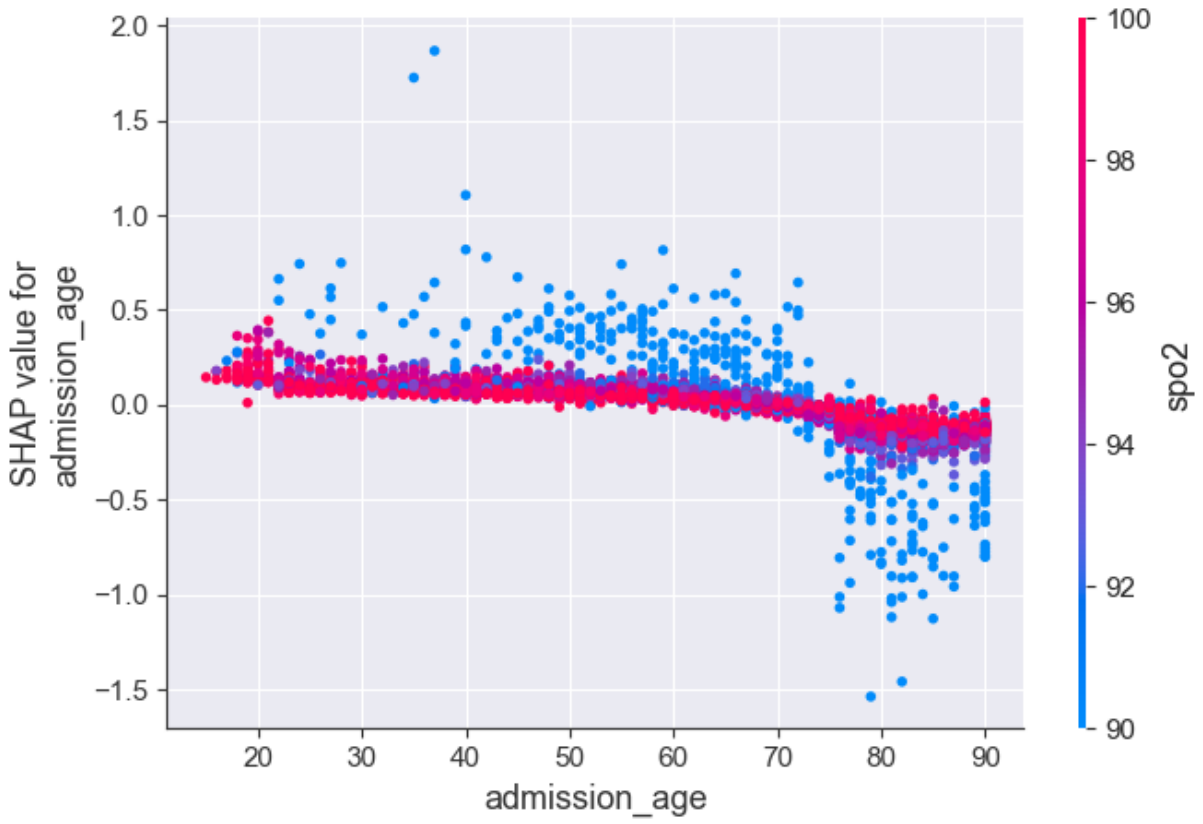
In [105…

```
#SpO2 and Age interaction
age_idx = list(X_test_shap_df.columns).index('admission_age')
shap.dependence_plot((spo2_idx, age_idx), shap_interaction_values, X_test_sh
```



The dependence between SpO2 and age was also explored. Similar to the two plots above, there is increased variability with the SHAP influence as the SpO2 values decrease. There is also a notable change in which direction the variability occurs at approximately age 70. Again, this matches clinically since people have lower oxygen carrying levels as they age.

In [106…
```
# Explore age
shap.dependence_plot(age_idx, shap_values, X_test_shap_df)
```



Appendix: Random Forest with SMOTER Oversampling

In [107…
```
# Combine X_train_imputed_df and y_train into single dataframe
# smogn.smoter expects a single dataframe features and target
train_data = X_train_imputed_df.copy()
train_data['gap'] = y_train
```

In [108…
```
#Apply SMOTER (time ~ 6 hr)
synthesized_data = smogn.smoter(
    data=train_data,
    y='gap',  # target column
    k=5,
    samp_method='extreme',
    rel_thres=0.3,
    rel_method='auto',
    rel_xtrm_type='high',
    rel_coef=0.8,
```

```
    rel_ctrl_pts_rg=None
)

# Extract features and target from synthesized data
X_train_smoter = synthesized_data.drop('gap', axis=1)
y_train_smoter = synthesized_data['gap']
```

```
dist_matrix: 100%|##########| 10203/10203 [6:07:44<00:00,  2.16s/it]
synth_matrix: 100%|##########| 10203/10203 [00:07<00:00, 1360.47it/s]
r_index: 100%|##########| 8162/8162 [00:05<00:00, 1363.67it/s]
```

In [109…
```
# Save X and y separately to reload if needed later
X_train_smoter.to_csv('X_train_smoter.csv', index=False)
y_train_smoter.to_csv('y_train_smoter.csv', index=False)
```

In [ ]:
```
# to load:
# X_train_smoter = pd.read_csv('X_train_smoter.csv')
# y_train_smoter = pd.read_csv('y_train_smoter.csv', squeeze=True)  # squeez
```

In [110…
```
X_train_smoter.shape
```

Out[110… (46011, 37)

In [111…
```
y_train_smoter.shape
```

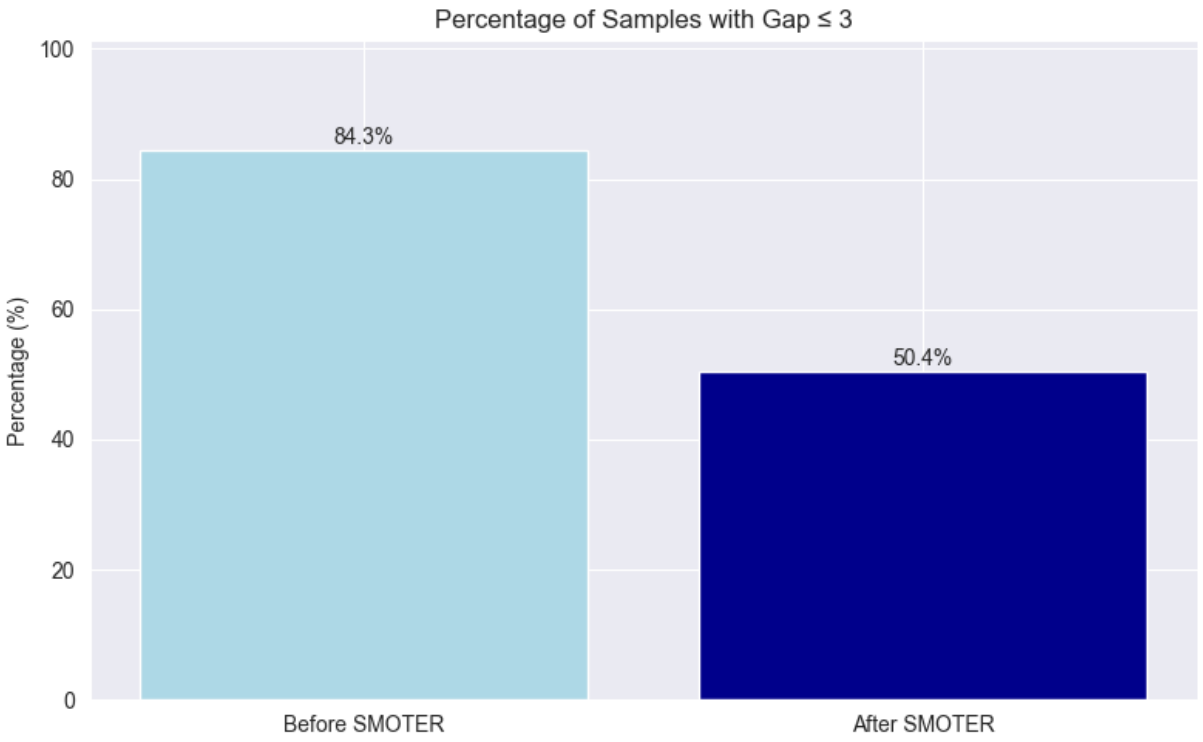
Out[111… (46011,)

In [113…
```
# Get percentage of samples with gap ≤ 3 (target value)
before_target_pct = (train_data['gap'] <= 3).mean() * 100
after_target_pct = (synthesized_data['gap'] <= 3).mean() * 100

# Print simple summary
print("Gap ≤ 3:")
print(f"  Before SMOTER: {before_target_pct:.2f}%")
print(f"  After SMOTER: {after_target_pct:.2f}%")
print(f"  Change: {after_target_pct - before_target_pct:.2f} percentage poin

# Simple comparison of data sizes
print("\nDataset Sizes:")
print(f"  Original training data: {len(train_data)} samples")
print(f"  After SMOTER: {len(synthesized_data)} samples")
print(f"  Change: {len(synthesized_data) - len(train_data)} samples")
```

```
Gap ≤ 3:
  Before SMOTER: 84.31%
  After SMOTER: 50.44%
  Change: -33.86 percentage points

Dataset Sizes:
  Original training data: 39274 samples
  After SMOTER: 46011 samples
  Change: 6737 samples
```

In [114…
```
# Data for visualization
labels = ['Before SMOTER', 'After SMOTER']
target_pct = [before_target_pct, after_target_pct]
```

```
# Create simple bar chart
plt.figure(figsize=(8, 5))
plt.bar(labels, target_pct, color=['lightblue', 'darkblue'])

# Add percentages on top of bars
for i, v in enumerate(target_pct):
    plt.text(i, v + 1, f'{v:.1f}%', ha='center')

# Add title and labels
plt.title('Percentage of Samples with Gap ≤ 3')
plt.ylabel('Percentage (%)')
plt.ylim(0, max(target_pct) * 1.2)  # Set y limit with some headroom

plt.tight_layout()
plt.show()
```



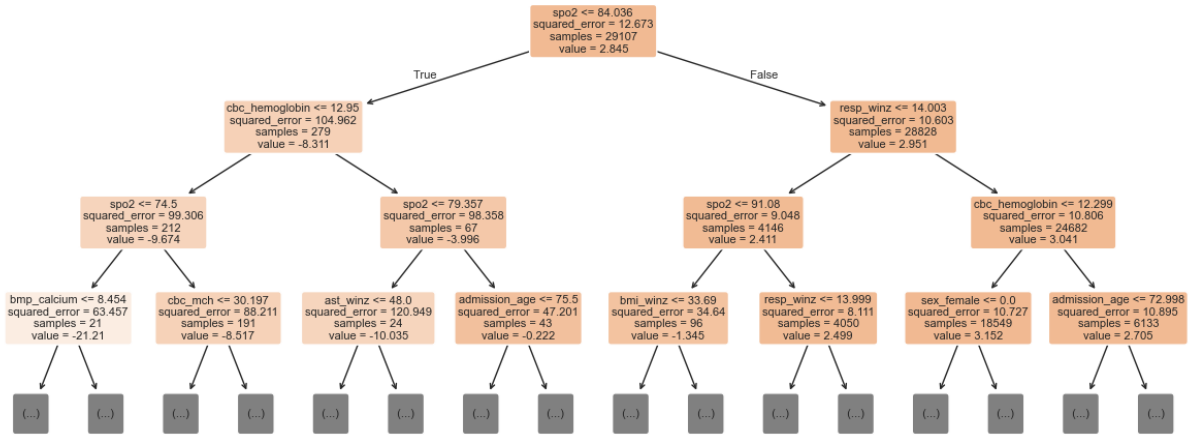
In [115…
```
# Initialize and build a Random forest model for SMOTER Data
rfr = RandomForestRegressor(
    n_estimators=100,
    max_features="sqrt",  # Square root of the number of features for a regr
    random_state=42,
    oob_score=True
)
rfr.fit(X_train_smoter, y_train_smoter)
y_pred_rfr = rfr.predict(X_test_imputed_df)
```

In [116…
```
# A visualization of the first tree
feature_names = X_train_smoter.columns.tolist()

# Extract the first tree from RFR model
first_tree = rfr.estimators_[0]
plt.figure(figsize=(15,6))
```

```
tree.plot_tree(first_tree,
               feature_names=feature_names,
               fontsize=8,
               filled=True,
               rounded=True,
               max_depth=3)
```

Out[116…

```
[Text(0.5, 0.9, 'spo2 <= 84.036\nsquared_error = 12.673\nsamples = 29107\nvalue = 2.845'),
 Text(0.25, 0.7, 'cbc_hemoglobin <= 12.95\nsquared_error = 104.962\nsamples = 279\nvalue = -8.311'),
 Text(0.375, 0.8, 'True  '),
 Text(0.125, 0.5, 'spo2 <= 74.5\nsquared_error = 99.306\nsamples = 212\nvalue = -9.674'),
 Text(0.0625, 0.3, 'bmp_calcium <= 8.454\nsquared_error = 63.457\nsamples = 21\nvalue = -21.21'),
 Text(0.03125, 0.1, '\n  (...)  \n'),
 Text(0.09375, 0.1, '\n  (...)  \n'),
 Text(0.1875, 0.3, 'cbc_mch <= 30.197\nsquared_error = 88.211\nsamples = 191\nvalue = -8.517'),
 Text(0.15625, 0.1, '\n  (...)  \n'),
 Text(0.21875, 0.1, '\n  (...)  \n'),
 Text(0.375, 0.5, 'spo2 <= 79.357\nsquared_error = 98.358\nsamples = 67\nvalue = -3.996'),
 Text(0.3125, 0.3, 'ast_winz <= 48.0\nsquared_error = 120.949\nsamples = 24\nvalue = -10.035'),
 Text(0.28125, 0.1, '\n  (...)  \n'),
 Text(0.34375, 0.1, '\n  (...)  \n'),
 Text(0.4375, 0.3, 'admission_age <= 75.5\nsquared_error = 47.201\nsamples = 43\nvalue = -0.222'),
 Text(0.40625, 0.1, '\n  (...)  \n'),
 Text(0.46875, 0.1, '\n  (...)  \n'),
 Text(0.75, 0.7, 'resp_winz <= 14.003\nsquared_error = 10.603\nsamples = 28828\nvalue = 2.951'),
 Text(0.625, 0.8, '  False'),
 Text(0.625, 0.5, 'spo2 <= 91.08\nsquared_error = 9.048\nsamples = 4146\nvalue = 2.411'),
 Text(0.5625, 0.3, 'bmi_winz <= 33.69\nsquared_error = 34.64\nsamples = 96\nvalue = -1.345'),
 Text(0.53125, 0.1, '\n  (...)  \n'),
 Text(0.59375, 0.1, '\n  (...)  \n'),
 Text(0.6875, 0.3, 'resp_winz <= 13.999\nsquared_error = 8.111\nsamples = 4050\nvalue = 2.499'),
 Text(0.65625, 0.1, '\n  (...)  \n'),
 Text(0.71875, 0.1, '\n  (...)  \n'),
 Text(0.875, 0.5, 'cbc_hemoglobin <= 12.299\nsquared_error = 10.806\nsamples = 24682\nvalue = 3.041'),
 Text(0.8125, 0.3, 'sex_female <= 0.0\nsquared_error = 10.727\nsamples = 18549\nvalue = 3.152'),
 Text(0.78125, 0.1, '\n  (...)  \n'),
 Text(0.84375, 0.1, '\n  (...)  \n'),
 Text(0.9375, 0.3, 'admission_age <= 72.998\nsquared_error = 10.895\nsamples = 6133\nvalue = 2.705'),
 Text(0.90625, 0.1, '\n  (...)  \n'),
 Text(0.96875, 0.1, '\n  (...)  \n')]
```



In [117…
```
#Evaluation metrics
mse = round(mean_squared_error(y_test, y_pred_rfr), 2)
rmse = round(np.sqrt(mean_squared_error(y_test, y_pred_rfr)), 2)  # Calculat

# View accuracy score
print("Mean Absolute Error:", round(mean_absolute_error(y_test, y_pred_rfr),
print("Mean Squared Error:", mse)
print("Root Mean Squared Error:", rmse)
print("R² Score:", round(r2_score(y_test, y_pred_rfr), 4))
```

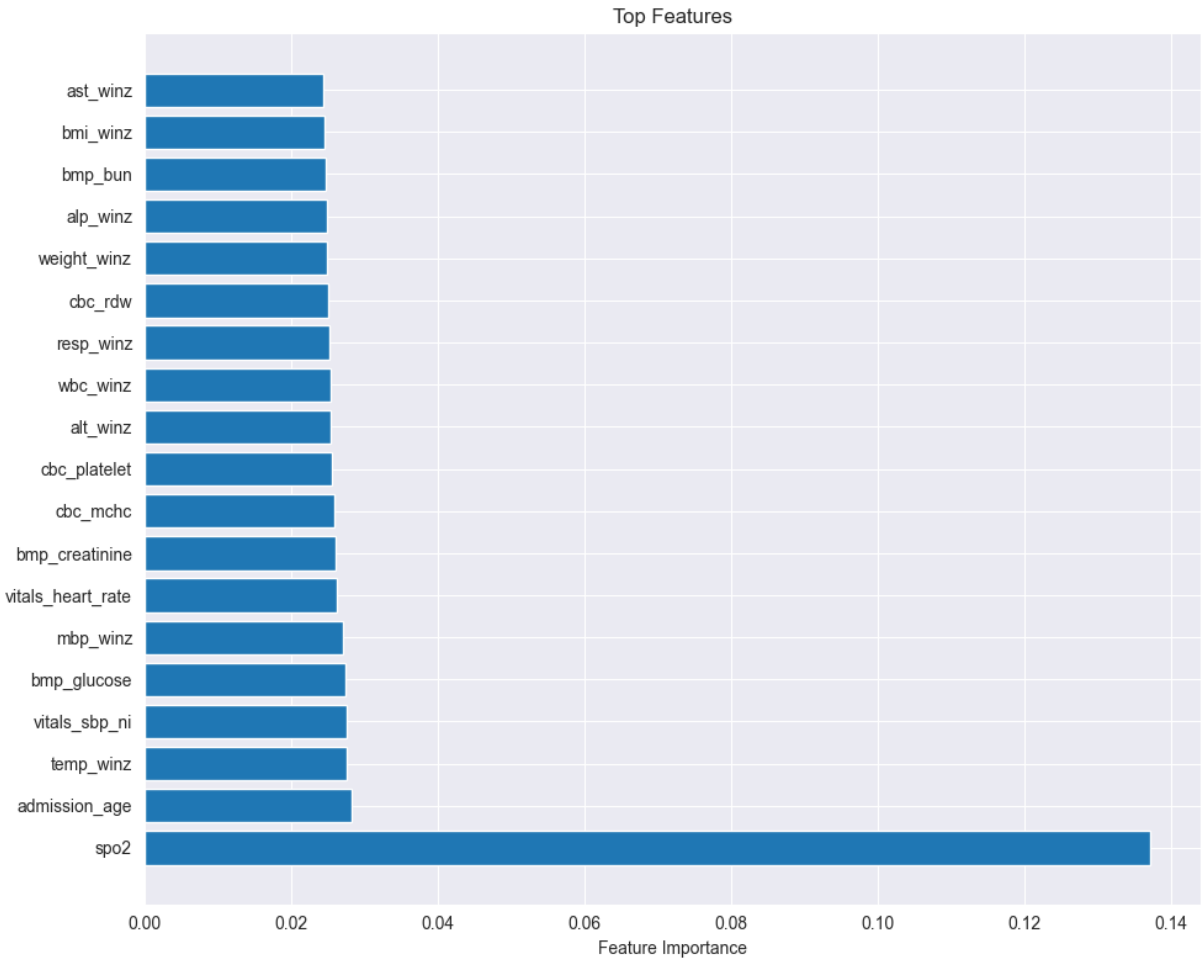
```
Mean Absolute Error: 3.14
Mean Squared Error: 16.79
Root Mean Squared Error: 4.1
R² Score: -0.1658
```

In [118…
```
# Select features using only training data to reduce data leakage
selector = SelectFromModel(rfr, threshold="median")
selector.fit(X_train_smoter, y_train_smoter)

# Get selected feature names
selected_features = X_train_smoter.columns[selector.get_support()]
print(f"Selected {len(selected_features)} features out of {X_train_smoter.sh
print(selected_features.tolist())

# Transform data to use only selected features
X_train_selected = selector.transform(X_train_smoter)
X_test_selected = selector.transform(X_test_imputed_df)
```

```
Selected 19 features out of 37
['admission_age', 'spo2', 'vitals_heart_rate', 'vitals_sbp_ni', 'cbc_mchc',
'cbc_platelet', 'cbc_rdw', 'bmp_bun', 'bmp_creatinine', 'bmp_glucose', 'weig
ht_winz', 'bmi_winz', 'resp_winz', 'mbp_winz', 'temp_winz', 'wbc_winz', 'alt
_winz', 'alp_winz', 'ast_winz']
```

In [119…
```
# Get feature importance
importances = rfr.feature_importances_
feature_names = X_train_smoter.columns
indices = np.argsort(importances)[::-1]

# Plot only top features
plt.figure(figsize=(10, 8))
```

```
plt.barh(range(19), importances[indices][:19])
plt.yticks(range(19), [feature_names[i] for i in indices[:19]])
plt.xlabel('Feature Importance')
plt.title('Top Features')
plt.tight_layout()
plt.show()
```



> **Note**: The output from this code is *tracking* of the process, *not an error*.

In [120…

```
# Define the objective function for Optuna (~ 10 hr 15 min)
def objective(trial):
    params = {
        'n_estimators': trial.suggest_int('n_estimators', 100, 800),
        'max_depth': trial.suggest_int('max_depth', 10, 80, log=True),
        'min_samples_split': trial.suggest_int('min_samples_split', 2, 10),
        'min_samples_leaf': trial.suggest_int('min_samples_leaf', 1, 4),
        'max_features': trial.suggest_categorical('max_features', ['sqrt', '
        'max_samples': trial.suggest_categorical('max_samples', [0.7, 0.9, N
    }

    rf = RandomForestRegressor(**params, random_state=42)
    score = cross_val_score(
        rf,
        X_train_selected,
        y_train_smoter,
```

```
        cv=3,
        scoring='neg_mean_squared_error',
        n_jobs=-1
    )
    # Convert MSE to RMSE
    return np.sqrt(-np.mean(score))  # Square root of MSE = RMSE

study = optuna.create_study(direction='minimize')
study.optimize(objective, n_trials=50)

print("Best trial:")
trial = study.best_trial
print("  Value: {}".format(trial.value))
print("  Params: ")
for key, value in trial.params.items():
    print("    {}: {}".format(key, value))
```

```
[I 2025-03-31 23:18:53,974] A new study created in memory with name: no-nam
e-2e19b5cf-271f-4140-b2a6-ca10130daebf
[I 2025-03-31 23:19:23,481] Trial 0 finished with value: 3.437134039861393 a
nd parameters: {'n_estimators': 264, 'max_depth': 10, 'min_samples_split': 6
, 'min_samples_leaf': 3, 'max_features': 0.5, 'max_samples': None}. Best is
trial 0 with value: 3.437134039861393.
[I 2025-03-31 23:19:42,088] Trial 1 finished with value: 3.3312001661434807
and parameters: {'n_estimators': 232, 'max_depth': 43, 'min_samples_split':
10, 'min_samples_leaf': 4, 'max_features': 'sqrt', 'max_samples': 0.7}. Best
is trial 1 with value: 3.3312001661434807.
[I 2025-03-31 23:21:08,485] Trial 2 finished with value: 3.2428630157702356
and parameters: {'n_estimators': 437, 'max_depth': 25, 'min_samples_split':
8, 'min_samples_leaf': 4, 'max_features': 0.5, 'max_samples': 0.9}. Best is
trial 2 with value: 3.2428630157702356.
[I 2025-03-31 23:22:08,652] Trial 3 finished with value: 3.3096582641204417
and parameters: {'n_estimators': 695, 'max_depth': 37, 'min_samples_split':
8, 'min_samples_leaf': 3, 'max_features': 'log2', 'max_samples': 0.7}. Best
is trial 2 with value: 3.2428630157702356.
[I 2025-03-31 23:23:12,752] Trial 4 finished with value: 3.282910891295772 a
nd parameters: {'n_estimators': 579, 'max_depth': 36, 'min_samples_split': 9
, 'min_samples_leaf': 3, 'max_features': 'sqrt', 'max_samples': None}. Best
is trial 2 with value: 3.2428630157702356.
[I 2025-03-31 23:26:44,455] Trial 5 finished with value: 3.195679089196417 a
nd parameters: {'n_estimators': 471, 'max_depth': 30, 'min_samples_split': 7
, 'min_samples_leaf': 3, 'max_features': None, 'max_samples': None}. Best is
trial 5 with value: 3.195679089196417.
[I 2025-03-31 23:27:23,481] Trial 6 finished with value: 3.2922673214817353
and parameters: {'n_estimators': 364, 'max_depth': 38, 'min_samples_split':
9, 'min_samples_leaf': 4, 'max_features': 'log2', 'max_samples': None}. Best
is trial 5 with value: 3.195679089196417.
[I 2025-03-31 23:32:48,003] Trial 7 finished with value: 3.211742019454139 a
nd parameters: {'n_estimators': 775, 'max_depth': 23, 'min_samples_split': 6
, 'min_samples_leaf': 4, 'max_features': None, 'max_samples': None}. Best is
trial 5 with value: 3.195679089196417.
[I 2025-03-31 23:39:02,777] Trial 8 finished with value: 3.2024510341000747
and parameters: {'n_estimators': 786, 'max_depth': 36, 'min_samples_split':
10, 'min_samples_leaf': 2, 'max_features': None, 'max_samples': None}. Best
is trial 5 with value: 3.195679089196417.
[I 2025-03-31 23:39:12,463] Trial 9 finished with value: 3.5123954539064144
and parameters: {'n_estimators': 221, 'max_depth': 10, 'min_samples_split':
5, 'min_samples_leaf': 2, 'max_features': 'log2', 'max_samples': 0.7}. Best
is trial 5 with value: 3.195679089196417.
[I 2025-03-31 23:44:08,187] Trial 10 finished with value: 3.2023204101798743
and parameters: {'n_estimators': 572, 'max_depth': 71, 'min_samples_split':
2, 'min_samples_leaf': 1, 'max_features': None, 'max_samples': 0.9}. Best is
trial 5 with value: 3.195679089196417.
[I 2025-03-31 23:48:33,774] Trial 11 finished with value: 3.202899605012628
and parameters: {'n_estimators': 544, 'max_depth': 79, 'min_samples_split':
2, 'min_samples_leaf': 1, 'max_features': None, 'max_samples': 0.9}. Best is
trial 5 with value: 3.195679089196417.
[I 2025-03-31 23:53:33,085] Trial 12 finished with value: 3.202106870282154
and parameters: {'n_estimators': 606, 'max_depth': 78, 'min_samples_split':
2, 'min_samples_leaf': 1, 'max_features': None, 'max_samples': 0.9}. Best is
trial 5 with value: 3.195679089196417.
[I 2025-03-31 23:54:07,687] Trial 13 finished with value: 3.289197041328712
and parameters: {'n_estimators': 111, 'max_depth': 15, 'min_samples_split':
```

3, 'min_samples_leaf': 2, 'max_features': None, 'max_samples': 0.9}. Best is trial 5 with value: 3.195679089196417.
[I 2025-04-01 00:32:10,870] Trial 14 finished with value: 3.206097915517393 and parameters: {'n_estimators': 481, 'max_depth': 57, 'min_samples_split': 4, 'min_samples_leaf': 1, 'max_features': None, 'max_samples': 0.9}. Best is trial 5 with value: 3.195679089196417.
[I 2025-04-01 01:07:44,316] Trial 15 finished with value: 3.2561439777081467 and parameters: {'n_estimators': 656, 'max_depth': 18, 'min_samples_split': 7, 'min_samples_leaf': 2, 'max_features': None, 'max_samples': None}. Best is trial 5 with value: 3.195679089196417.
[I 2025-04-01 01:42:40,660] Trial 16 finished with value: 3.191185274731371 and parameters: {'n_estimators': 433, 'max_depth': 56, 'min_samples_split': 4, 'min_samples_leaf': 3, 'max_features': None, 'max_samples': 0.9}. Best is trial 16 with value: 3.191185274731371.
[I 2025-04-01 01:43:25,483] Trial 17 finished with value: 3.2660513884031195 and parameters: {'n_estimators': 367, 'max_depth': 54, 'min_samples_split': 4, 'min_samples_leaf': 3, 'max_features': 'sqrt', 'max_samples': None}. Best is trial 16 with value: 3.191185274731371.
[I 2025-04-01 01:45:04,351] Trial 18 finished with value: 3.2086858568845424 and parameters: {'n_estimators': 389, 'max_depth': 53, 'min_samples_split': 5, 'min_samples_leaf': 3, 'max_features': 0.5, 'max_samples': 0.9}. Best is trial 16 with value: 3.191185274731371.
[I 2025-04-01 02:04:30,302] Trial 19 finished with value: 3.2083953859695935 and parameters: {'n_estimators': 489, 'max_depth': 29, 'min_samples_split': 7, 'min_samples_leaf': 3, 'max_features': None, 'max_samples': 0.7}. Best is trial 16 with value: 3.191185274731371.
[I 2025-04-01 02:22:57,373] Trial 20 finished with value: 3.244148915455337 and parameters: {'n_estimators': 288, 'max_depth': 19, 'min_samples_split': 4, 'min_samples_leaf': 2, 'max_features': None, 'max_samples': None}. Best is trial 16 with value: 3.191185274731371.
[I 2025-04-01 02:56:10,785] Trial 21 finished with value: 3.1900342444517187 and parameters: {'n_estimators': 652, 'max_depth': 66, 'min_samples_split': 3, 'min_samples_leaf': 3, 'max_features': None, 'max_samples': 0.9}. Best is trial 21 with value: 3.1900342444517187.
[I 2025-04-01 03:16:55,701] Trial 22 finished with value: 3.189698996534239 and parameters: {'n_estimators': 666, 'max_depth': 62, 'min_samples_split': 3, 'min_samples_leaf': 3, 'max_features': None, 'max_samples': 0.9}. Best is trial 22 with value: 3.189698996534239.
[I 2025-04-01 04:25:29,063] Trial 23 finished with value: 3.1899063187826204 and parameters: {'n_estimators': 704, 'max_depth': 62, 'min_samples_split': 3, 'min_samples_leaf': 3, 'max_features': None, 'max_samples': 0.9}. Best is trial 22 with value: 3.189698996534239.
[I 2025-04-01 05:41:40,896] Trial 24 finished with value: 3.1956823197295363 and parameters: {'n_estimators': 718, 'max_depth': 65, 'min_samples_split': 3, 'min_samples_leaf': 4, 'max_features': None, 'max_samples': 0.9}. Best is trial 22 with value: 3.189698996534239.
[I 2025-04-01 06:19:45,447] Trial 25 finished with value: 3.1899519877667255 and parameters: {'n_estimators': 658, 'max_depth': 49, 'min_samples_split': 3, 'min_samples_leaf': 3, 'max_features': None, 'max_samples': 0.9}. Best is trial 22 with value: 3.189698996534239.
[I 2025-04-01 06:21:18,267] Trial 26 finished with value: 3.248064907527337 and parameters: {'n_estimators': 731, 'max_depth': 45, 'min_samples_split': 3, 'min_samples_leaf': 2, 'max_features': 'log2', 'max_samples': 0.9}. Best is trial 22 with value: 3.189698996534239.
[I 2025-04-01 06:55:54,854] Trial 27 finished with value: 3.2069383091447103 and parameters: {'n_estimators': 651, 'max_depth': 47, 'min_samples_split':

5, 'min_samples_leaf': 3, 'max_features': 0.5, 'max_samples': 0.9}. Best is trial 22 with value: 3.189698996534239.
[I 2025-04-01 06:56:54,860] Trial 28 finished with value: 3.296510020293107 and parameters: {'n_estimators': 528, 'max_depth': 63, 'min_samples_split': 2, 'min_samples_leaf': 4, 'max_features': 'sqrt', 'max_samples': 0.9}. Best is trial 22 with value: 3.189698996534239.
[I 2025-04-01 07:00:16,841] Trial 29 finished with value: 3.2071107885201195 and parameters: {'n_estimators': 621, 'max_depth': 48, 'min_samples_split': 3, 'min_samples_leaf': 3, 'max_features': 0.5, 'max_samples': 0.9}. Best is trial 22 with value: 3.189698996534239.
[I 2025-04-01 07:32:28,805] Trial 30 finished with value: 3.1899356705427793 and parameters: {'n_estimators': 749, 'max_depth': 41, 'min_samples_split': 5, 'min_samples_leaf': 3, 'max_features': None, 'max_samples': 0.9}. Best is trial 22 with value: 3.189698996534239.
[I 2025-04-01 08:17:34,566] Trial 31 finished with value: 3.1897868291145737 and parameters: {'n_estimators': 750, 'max_depth': 50, 'min_samples_split': 5, 'min_samples_leaf': 3, 'max_features': None, 'max_samples': 0.9}. Best is trial 22 with value: 3.189698996534239.
[I 2025-04-01 08:22:42,693] Trial 32 finished with value: 3.1898385805162777 and parameters: {'n_estimators': 745, 'max_depth': 42, 'min_samples_split': 5, 'min_samples_leaf': 3, 'max_features': None, 'max_samples': 0.9}. Best is trial 22 with value: 3.189698996534239.
[I 2025-04-01 08:27:21,975] Trial 33 finished with value: 3.197248033756243 and parameters: {'n_estimators': 695, 'max_depth': 32, 'min_samples_split': 6, 'min_samples_leaf': 4, 'max_features': None, 'max_samples': 0.9}. Best is trial 22 with value: 3.189698996534239.
[I 2025-04-01 08:32:06,344] Trial 34 finished with value: 3.199698207421491 and parameters: {'n_estimators': 798, 'max_depth': 57, 'min_samples_split': 4, 'min_samples_leaf': 3, 'max_features': None, 'max_samples': 0.7}. Best is trial 22 with value: 3.189698996534239.
[I 2025-04-01 08:33:19,221] Trial 35 finished with value: 3.273509217560522 and parameters: {'n_estimators': 692, 'max_depth': 69, 'min_samples_split': 6, 'min_samples_leaf': 3, 'max_features': 'sqrt', 'max_samples': 0.9}. Best is trial 22 with value: 3.189698996534239.
[I 2025-04-01 08:38:20,919] Trial 36 finished with value: 3.1955207937722974 and parameters: {'n_estimators': 744, 'max_depth': 42, 'min_samples_split': 5, 'min_samples_leaf': 4, 'max_features': None, 'max_samples': 0.9}. Best is trial 22 with value: 3.189698996534239.
[I 2025-04-01 08:39:21,700] Trial 37 finished with value: 3.2950110333518574 and parameters: {'n_estimators': 698, 'max_depth': 61, 'min_samples_split': 4, 'min_samples_leaf': 3, 'max_features': 'log2', 'max_samples': 0.7}. Best is trial 22 with value: 3.189698996534239.
[I 2025-04-01 08:41:35,340] Trial 38 finished with value: 3.213573957344516 and parameters: {'n_estimators': 610, 'max_depth': 32, 'min_samples_split': 7, 'min_samples_leaf': 2, 'max_features': 0.5, 'max_samples': 0.9}. Best is trial 22 with value: 3.189698996534239.
[I 2025-04-01 08:46:59,319] Trial 39 finished with value: 3.1901705095239405 and parameters: {'n_estimators': 758, 'max_depth': 39, 'min_samples_split': 6, 'min_samples_leaf': 3, 'max_features': None, 'max_samples': 0.9}. Best is trial 22 with value: 3.189698996534239.
[I 2025-04-01 08:48:14,640] Trial 40 finished with value: 3.3172899537577534 and parameters: {'n_estimators': 798, 'max_depth': 26, 'min_samples_split': 8, 'min_samples_leaf': 4, 'max_features': 'sqrt', 'max_samples': 0.9}. Best is trial 22 with value: 3.189698996534239.
[I 2025-04-01 08:53:36,680] Trial 41 finished with value: 3.1899021851164147 and parameters: {'n_estimators': 754, 'max_depth': 51, 'min_samples_split':

```
5, 'min_samples_leaf': 3, 'max_features': None, 'max_samples': 0.9}. Best is
trial 22 with value: 3.189698996534239.
[I 2025-04-01 08:58:43,097] Trial 42 finished with value: 3.1898319003829796
and parameters: {'n_estimators': 713, 'max_depth': 50, 'min_samples_split':
5, 'min_samples_leaf': 3, 'max_features': None, 'max_samples': 0.9}. Best is
trial 22 with value: 3.189698996534239.
[I 2025-04-01 09:04:08,506] Trial 43 finished with value: 3.1897563590503877
and parameters: {'n_estimators': 764, 'max_depth': 49, 'min_samples_split':
5, 'min_samples_leaf': 3, 'max_features': None, 'max_samples': 0.9}. Best is
trial 22 with value: 3.189698996534239.
[I 2025-04-01 09:08:59,156] Trial 44 finished with value: 3.190866681109328
and parameters: {'n_estimators': 677, 'max_depth': 34, 'min_samples_split':
6, 'min_samples_leaf': 3, 'max_features': None, 'max_samples': 0.9}. Best is
trial 22 with value: 3.189698996534239.
[I 2025-04-01 09:13:37,224] Trial 45 finished with value: 3.2003584704756602
and parameters: {'n_estimators': 725, 'max_depth': 45, 'min_samples_split':
5, 'min_samples_leaf': 2, 'max_features': None, 'max_samples': 0.7}. Best is
trial 22 with value: 3.189698996534239.
[I 2025-04-01 09:15:27,195] Trial 46 finished with value: 3.274003427068744
and parameters: {'n_estimators': 770, 'max_depth': 38, 'min_samples_split':
5, 'min_samples_leaf': 3, 'max_features': 'log2', 'max_samples': 0.9}. Best
is trial 22 with value: 3.189698996534239.
[I 2025-04-01 09:22:14,871] Trial 47 finished with value: 3.189528794222912
and parameters: {'n_estimators': 775, 'max_depth': 74, 'min_samples_split':
6, 'min_samples_leaf': 3, 'max_features': None, 'max_samples': 0.9}. Best is
trial 47 with value: 3.189528794222912.
[I 2025-04-01 09:26:12,184] Trial 48 finished with value: 3.195954495594467
and parameters: {'n_estimators': 572, 'max_depth': 71, 'min_samples_split':
7, 'min_samples_leaf': 4, 'max_features': None, 'max_samples': 0.9}. Best is
trial 47 with value: 3.189528794222912.
[I 2025-04-01 09:32:01,141] Trial 49 finished with value: 3.1991102565814002
and parameters: {'n_estimators': 779, 'max_depth': 73, 'min_samples_split':
8, 'min_samples_leaf': 2, 'max_features': None, 'max_samples': 0.9}. Best is
trial 47 with value: 3.189528794222912.
Best trial:
  Value: 3.189528794222912
  Params:
    n_estimators: 775
    max_depth: 74
    min_samples_split: 6
    min_samples_leaf: 3
    max_features: None
    max_samples: 0.9
```

In [121…

```python
# Save the completed study
joblib.dump(study, 'optuna_study.pkl')

# To load it later:
# loaded_study = joblib.load('optuna_study.pkl')
```

Out[121…

```
['optuna_study.pkl']
```

In [122…

```python
# Train the best model on the full training set
best_params = study.best_params
best_rf = RandomForestRegressor(**best_params, random_state=42)
best_rf.fit(X_train_selected, y_train_smoter)
```

Out[122…

```
▾                    RandomForestRegressor
RandomForestRegressor(max_depth=74, max_features=None, max_samples=
0.9,
                      min_samples_leaf=3, min_samples_split=6, n_es
timators=775,
                      random_state=42)
```

In [123…

```
# Make predictions
y_pred_optuna = best_rf.predict(X_test_selected)
```

In [124…

```
#Evaluation metrics
mse_optuna = round(mean_squared_error(y_test, y_pred_optuna), 2)
rmse_optuna = round(np.sqrt(mean_squared_error(y_test, y_pred_optuna)), 2)
# View accuracy scores
print("Mean Absolute Error:", round(mean_absolute_error(y_test, y_pred_optun
print("Mean Squared Error:", mse_optuna)
print("Root Mean Squared Error:", rmse_optuna)
print("R² Score:", round(r2_score(y_test, y_pred_optuna), 4))
```

```
Mean Absolute Error: 2.38
Mean Squared Error: 12.08
Root Mean Squared Error: 3.48
R² Score: 0.1614
```

Comparison of SMOTER basic and final model

Metric	Initial	Final
Mean Absolute Error	3.14	2.38
Mean Squared Error	16.79	12.08
Root Mean Squared Error	4.1	3.48
R^2 Score	-0.1658	0.1614

Return to Table of Contents

F: Sources

Brownlee, J. (2020, August 15). *Data Leakage in Machine Learning*. Machine Learning Mastery. Retrieved February 15, 2025, from https://machinelearningmastery.com/data-leakage-machine-learning/

Grippi, M. A., MD (2020, December 4). *How to calculate the total oxygen content of arterial blood*. MedMastery. Retrieved February 15, 2025, from https://www.medmastery.com/guides/blood-gas-analysis-clinical-guide/how-calculate-total-oxygen-content-arterial-blood

Horsch, A. (2021, February 15). *Detecting and Treating Outliers In Python — Part 3*. Towards Data Science. Retrieved February 1, 2025, from https://towardsdatascience.com/detecting-and-treating-outliers-in-python-part-3-dcb54abaf7b0/

Kunz, N. (2020). *SMOGN: Synthetic Minority Over-Sampling Technique for Regression with Gaussian Noise (Version 0.1.2)* [Computer software]. PyPI. Retrieved February 20, 2025, from https://pypi.org/project/smogn/

Preferred Networks, Inc. (2024). *Optuna - A hyperparameter optimization framework*. Optuna. Retrieved February 15, 2025, from https://optuna.org/

McKinney, W. (2022). *Python for Data Analysis* (3rd ed.). O'Reilly.

Molnar, C. (2025). *Interpretable machine learning: A guide for making black box models explainable* (3rd ed.). Retrieved March 15, 2025 from https://christophm.github.io/interpretable-ml-book/

National Institutes of Health, National Heart, Lung, and Blood Institute. (2025). ARDSNet Protocol Card. Retrieved February 15, 2025, from https://icu.smhs.gwu.edu/sites/g/files/zaskib1161/files/2025-01/ardsnet_protocol_card.pdf

Roepke , B. (2024, March 5). *Don't Get Caught in the Trap of Imbalanced Data When Building a Model*. Data Knows All. Retrieved February 20, 2025, from https://dataknowsall.com/blog/imbalanced.html

Sjoding, M. W., Iwashyna, T. J., & Valley, T. S. (2023). *Change the framework for pulse oximeter regulation to ensure clinicians can give patients the oxygen they need*. American journal of respiratory and critical care medicine, 207(6), 661-664.

Varshney, P. (2023, February 28). *Q-Q Plots Explained*. BuiltIn. Retrieved February 25, 2025, from https://builtin.com/data-science/q-q-plot

Wong AI, Charpignon M, Kim H, et al. *Analysis of Discrepancies Between Pulse Oximetry and Arterial Oxygen Saturation Measurements by Race and Ethnicity and Association With Organ Dysfunction and Mortality*. JAMA Netw Open. 2021;4(11):e2131674. doi:10.1001/jamanetworkopen.2021.31674

In [ ]: