Predicting ICU Patient Clinical Deterioration - Capstone Project Report

Ashley Kirchner

Background

Intensive care units (ICU) provide critical care services to patients who have life-threatening conditions or are at risk for severe complications. The focus is on real-time monitoring, rapid intervention, and stabilization of patients with complex medical conditions. ICU patients require continuous monitoring of vital signs, laboratory results, and other data. (Ref: ChatGPT)

Early detection of clinical deterioration in ICU patients is crucial for improving patient outcomes and reducing mortality. Traditional methods rely on periodic manual assessments by healthcare professionals, which may be subjective and prone to delays in recognizing critical changes in patient conditions. Machine learning models can analyze vast amounts of real-time data from various sources, including vital signs, laboratory results, and electronic health records, to identify patterns and predict the onset of clinical deterioration with high accuracy. These models can alert healthcare teams to potential complications before they become apparent to human observers, enabling timely interventions and potentially preventing adverse events. The ability to predict clinical deterioration in ICU patients can lead to more proactive and personalized care, optimizing resource allocation and improving overall patient outcomes in critical care settings. (Ref: ChatGPT)

Objective

Participating in this competition will contribute to Kaggle's mission to "Help the world learn data science." Specifically, the Kaggle team hopes to "Develop a model that accurately predicts the onset of clinical deterioration in ICU patients, enabling timely interventions and improving patient outcomes." (Ref: Kaggle Competition Description)

Data Source

The data for this competition is provided by Johns Hopkins University and Stanford University. The PI and lead for the Computational Physiology lab is Leo Anthony Celi. The data is a subset of the MIMIC-IV database, which is a large, deidentified dataset of patients admitted to the intensive care unit at a medical center. The data includes various medical records, including vital signs, laboratory results, and other clinical measurements. The data is stored in Parquet files, which are a columnar storage format optimized for efficient data processing and retrieval. The data is organized into multiple files, with each file containing information for a specific patient group or time period. This structure allows for efficient loading and processing of relevant subsets of the data for analysis and modeling. (Ref: ChatGPT)

Model Variables

Vital signs: Heart rate, blood pressure, temperature, oxygen saturation, respiratory rate, etc. Laboratory results: Blood cell counts, electrolytes, kidney function tests, liver function tests, etc. Medications: Administered drugs, dosages, and timing. Clinical notes: Physician and nurse notes, diagnoses, procedures, etc. (Ref: ChatGPT)

The competition organizers provided a data dictionary (dictionary.csv) that lists the variables and their descriptions. These variables include various medical measurements, laboratory results, medications, and other clinical data. The data dictionary provides a comprehensive overview of the available features and their definitions, which is crucial for understanding the data and developing effective models. (Ref: ChatGPT)

The data is organized into several directories, including:

- train_timeseries: Contains time-series data for patients in the training set.
- test_timeseries: Contains time-series data for patients in the test set.
- train_static.parquet: Contains static data (e.g., demographics, publication info) for patients in the training set.
- test_static.parquet: Contains static data (e.g., demographics, publication info) for patients in the test set.
- dictionary.csv: Provides descriptions of the variables in the data.

Patient Static Data

train_static.parquet: 1.6 MB, 2,000 patient records, shape: (2000, 10)
test_static.parquet: 1.6 MB, 2,000 patient records, shape: (2000, 10)

The static data includes information about patients, such as demographics and publication information. The data is stored in Parquet files, which are a columnar storage format optimized for efficient data processing and retrieval. The data is organized into multiple files, with each file containing information for a specific patient group or time period. This structure allows for efficient loading and processing of relevant subsets of the data for analysis and modeling. (Ref: ChatGPT)

Patient Time Series Data

train_timeseries: 1.7 GB, 2,000 patient records, shape: (2000, 1000, 19)
test_timeseries: 1.7 GB, 2,000 patient records, shape: (2000, 1000, 19)

The time-series data includes various medical measurements taken over time, such as vital signs and laboratory results. The data is stored in Parquet files, which are a columnar storage format optimized for efficient data processing and retrieval. The data is organized into multiple files, with each file containing information for a specific patient group or time period. This structure allows for efficient loading and processing of relevant subsets of the data for analysis and modeling. (Ref: ChatGPT)

DATA FIELDS (for train_static.parquet and test_static.parquet)

- patient_id: A unique identifier for each patient.
- hospital_id: The identifier for the hospital where the patient was admitted.
- gender: The patient's gender.
- age: The patient's age.
- ethnicity: The patient's ethnicity.
- weight: The patient's weight.
- height: The patient's height.
- apache_4a_hospital_death_prob: The APACHE IV hospital death probability.
- apache_4a_icu_death_prob: The APACHE IV ICU death probability.
- gender_imputed: Indicates if the gender was imputed.
- ethnicity_imputed: Indicates if the ethnicity was imputed.

DATA FIELDS (for train_timeseries and test_timeseries)

- patient_id: A unique identifier for each patient.
- time_stamp: The time stamp for the vital sign measurement, in minutes from the start of the ICU stay.
- heart_rate: The patient's heart rate.
- sbp: The patient's systolic blood pressure.
- dbp: The patient's diastolic blood pressure.
- mbp: The patient's mean blood pressure.
- resp_rate: The patient's respiratory rate.
- spo2: The patient's oxygen saturation.
- temperature: The patient's temperature.
- urine_output_rate: The patient's urine output rate.
- mechanical_ventilation: Indicates if the patient was on mechanical ventilation.
- vasopressor_indicator: Indicates if the patient was on vasopressors.
- inotrope_indicator: Indicates if the patient was on inotropes.
- lactate: The patient's lactate level.
- creatinine: The patient's creatinine level.
- bilirubin_total: The patient's total bilirubin level.
- platelets: The patient's platelet count.
- wbc: The patient's white blood cell count.
- hematocrit: The patient's hematocrit.
- sodium: The patient's sodium level.
- potassium: The patient's potassium level.
- glucose: The patient's glucose level.
- bun: The patient's blood urea nitrogen level.

Data Frequency / Gaps

The variable "time_stamp" in the train/test timeseries files represents minutes from the start of the ICU stay. The data frequency varies, with some variables measured more frequently than others. The data may contain gaps or missing values, which need to be addressed during data preprocessing. The data is not uniformly sampled, and the time intervals between measurements can vary. The data may also contain outliers or erroneous values, which need to be identified and handled appropriately. (Ref: ChatGPT)

Data Imbalance

The data is highly imbalanced, with only a small percentage of patients experiencing clinical deterioration. This imbalance is a significant challenge in developing accurate predictive models. The imbalance can lead to models that are biased toward the majority class (non-deterioration) and have poor performance in predicting the minority class (deterioration). To address this imbalance, various techniques can be employed, such as oversampling the minority class, undersampling the majority class, or using cost-sensitive learning algorithms. The choice of the most appropriate technique depends on the specific characteristics of the data and the goals of the modeling. (Ref: ChatGPT)

Data Dictionary

The data dictionary (dictionary.csv) provides a comprehensive overview of the available features and their definitions. It includes information about the variable name, description, data type, and other relevant information. The data dictionary is crucial for understanding the data and developing effective models. It provides a clear and concise reference for the meaning of each variable and its relevance to the problem of predicting clinical deterioration. The data dictionary should be consulted frequently throughout the data exploration and modeling process to ensure a thorough understanding of the data and its implications. (Ref: ChatGPT)

Patient Static Data

train_static.parquet: 1.6 MB, 2,000 patient records, shape: (2000, 10)

```
$ pip install -q -U kaggle
$ mkdir ~/.kaggle
$ cp kaggle.json ~/.kaggle/
$ ls ~/.kaggle
$ chmod 600 /home/jupyter/.kaggle/kaggle.json
$ kaggle competitions download -c physionet-ai-clinical-monitoring
$ ls *.zip
$ unzip -q physionet-ai-clinical-monitoring.zip -d data
$ ls -lh data/
$ ls -lh data/train_timeseries/ | head -5
$ ls -lh data/test_timeseries/ | head -5
$ head -c 100 data/dictionary.csv
$ wc -l data/dictionary.csv
$ head -c 500 data/train_static.parquet | od -c | head -20
$ pip install -q pandas pyarrow numpy matplotlib seaborn scikit-learn
$ python -c "import pandas, pyarrow, numpy, matplotlib, seaborn, sklearn; print('All packages installed successfully')"
$ python -c "import pandas; print(pandas.__version__)"
$ python -c "import numpy; print(numpy.__version__)"
$ python -c "import matplotlib; print(matplotlib.__version__)"
$ python -c "import seaborn; print(seaborn.__version__)"
$ python -c "import sklearn; print(sklearn.__version__)"
$ pip freeze > requirements.txt
$ head -20 requirements.txt
$ ls -lh requirements.txt
$ ls -lh data/train_static.parquet data/test_static.parquet
$ ls -lh data/train_timeseries/ data/test_timeseries/ | head -10
```

```
$ ls -lh data/train_static.parquet
-rw-r--r-- 1 jupyter jupyter 1.6M Jun 11 19:44 data/train_static.parquet
$ head -c 500 data/train_static.parquet | od -c | head -20
0000000   P   A   R   1   l   s   c   h   e   m   a   L   e   n   g   t
0000020   h   =   2   0   0   0   0   0   0   0   0   0   0   0   0   0
0000040   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0
0000060   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0
0000100   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0
0000120   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0
0000140   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0
0000160   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0
0000200   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0
0000220   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0
0000240   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0
0000260   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0
0000300   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0
0000320   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0
0000340   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0
0000360   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0
0000400   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0
0000420   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0
0000440   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0
0000460   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0
0000500   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0
```

```
$ head -c 100 data/dictionary.csv
variable,description,type,source
$ wc -l data/dictionary.csv
26 data/dictionary.csv
$ head -c 500 data/train_static.parquet | od -c | head -20
0000000   P   A   R   1   l   s   c   h   e   m   a   L   e   n   g   t
0000020   h   =   2   0   0   0   0   0   0   0   0   0   0   0   0   0
0000040   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0
0000060   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0
0000100   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0
0000120   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0
0000140   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0
0000160   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0
0000200   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0
0000220   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0
0000240   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0
0000260   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0
0000300   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0
0000320   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0
0000340   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0
0000360   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0
0000400   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0
0000420   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0
0000440   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0
0000460   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0
0000500   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0
```

```
$ pip install -q pandas pyarrow numpy matplotlib seaborn scikit-learn
$ python -c "import pandas, pyarrow, numpy, matplotlib, seaborn, sklearn; print('All packages installed successfully')"
$ python -c "import pandas; print(pandas.__version__)"
$ python -c "import numpy; print(numpy.__version__)"
$ python -c "import matplotlib; print(matplotlib.__version__)"
$ python -c "import seaborn; print(seaborn.__version__)"
$ python -c "import sklearn; print(sklearn.__version__)"
$ pip freeze > requirements.txt
$ head -20 requirements.txt
$ ls -lh requirements.txt
$ ls -lh data/train_static.parquet data/test_static.parquet
$ ls -lh data/train_timeseries/ data/test_timeseries/ | head -10
```

```
$ head -c 500 data/train_static.parquet | od -c | head -20
0000000   P   A   R   1   l   s   c   h   e   m   a   L   e   n   g   t
0000020   h   =   2   0   0   0   0   0   0   0   0   0   0   0   0   0
0000040   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0
0000060   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0
0000100   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0
0000120   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0
0000140   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0
0000160   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0
0000200   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0
0000220   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0
0000240   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0
0000260   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0
0000300   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0
0000320   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0
0000340   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0
0000360   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0
0000400   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0
0000420   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0
0000440   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0
0000460   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0
0000500   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0
```

Inferential Statistics - Frequentist

In [57]:
```
# perform hypothesis test of spas2 score between those passed away at hospital and survived

from scipy.stats import ttest_ind
ttest_name_list = ['admission', 'ud', 'bun', 'Bicarbonate', 'ventilation', 'Temp', 'Bilirubin', 'gcs', 'AGE', 'UO',
                   'saps2', 'Potassium_0.0', 'Potassium_3.0', 'Sodium_0.0', 'Sodium_1.0', 'Sodium_5.0', 'WBC_0.0',
                   'WBC_3.0', 'hr_0.0', 'hr_2.0', 'hr_4.0', 'hr_7.0', 'hr_11.0', 'bp_0.0', 'bp_2.0', 'bp_5.0', 'bp_13.
0']
for i in (ttest_name_list):
    stat, pvalue=ttest_ind(saps[i], saps['hdeath'])
    print(i, stat, pvalue)
```

```
admission 422.0928311045037 0.0
ud 79.48553209547315 0.0
bun 103.82321908155082 0.0
Bicarbonate 50.202907263333376 0.0
ventilation 269.5616326641353 0.0
Temp 428.78581356046413 0.0
Bilirubin 68.31563939515497 0.0
gcs 185.50528236030465 0.0
AGE 374.9833756789964 0.0
UO -86.57830736617781 0.0
saps2 528.4388303796287 0.0
Potassium_0.0 613.9450781425617 0.0
Potassium_3.0 -59.82832139042311 0.0
Sodium_0.0 575.893954295901 0.0
Sodium_1.0 -64.01776903987019 0.0
Sodium_5.0 -66.93879118526705 0.0
WBC_0.0 572.7364948958383 0.0
WBC_3.0 -47.051164834922716 0.0
hr_0.0 137.92767245017222 0.0
hr_2.0 136.97522688863378 0.0
hr_4.0 -27.759615489236438 4.484733069363873e-169
hr_7.0 -79.36376090302323 0.0
hr_11.0 -31.431895017712954 5.427222076346614e-216
bp_0.0 33.35969234249016 6.556038544175584e-243
bp_2.0 -70.68488389053853 0.0
bp_5.0 162.14385989281462 0.0
bp_13.0 90.75917162078173 0.0
```

Indipendent ttest result shows:

- There is a statistically significant diffrence in admission type between deceased and survived ICU patients `(tstat=422, pvalue=0.0)`. On average we have higher number of unscheduled surgical admissions among deceased ICU patients.
- There is a statistically significant diffrence in the diagnosis of chronic disease between deceased and survived ICU patients `(tstat=79.5, pvalue=0.0)`. On average we have higher number of ICU patients with chronic dieases such as metastatic cancer,hematologic malignancy, and AIDS when compared between decieased and survived ICU stay.
- There is a statistically significant diffrence in blood urea nitrogin levels between deceased and survived ICU patients `(tstat=103.8, pvalue=0.0)`. On average we have higher number of ICU patients with high levels of blood urea nitrogen among deceased ICU patients.
- There is a statistically significant diffrence in blood bicarbonate levels between deceased and survived ICU patients `(tstat=50, pvalue=0.0)`. On average we have higher number of ICU patients with low levels of blood bicarbonate levels among deceased ICU patients.
- There is a statistically significant diffrence in ventilation use between deceased and survived ICU patients `(tstat=269.6, pvalue=0.0)`. On average we have higher number of ICU patients on ventilation among deceased ICU patients.
- There is a statistically significant diffrence in level of body temprature between deceased and survived ICU patients `(tstat=428.8, pvalue=0.0)`. On average we have patients with high body temprature among deceased ICU patients.
- There is a statistically significant diffrence in Bilirubin level between deceased and survived ICU patients `(tstat=68.3, pvalue=0.0)`. On average we have higher number of patients with high Bilirubin levels among deceased ICU patients.
- There is a statistically significant diffrence total Glasgow Coma Score between deceased and survived ICU patients `(tstat=185.5, pvalue=0.0)`. On average we have higher number of ICU patients with higher Glsgow Coma Score among deceased ICU patients. The total Glasgow Coma Score is based on best eye response, best verbal response, and best motor response scores.
- There is a statistically significant diffrence in patient age between deceased and survived ICU patients `(tstat=375, pvalue=0.0)`. On average we have higher proportion of deceased ICU patients who are older.
- There is a statistically significant diffrence in urine output between deceased and survived ICU patients `(tstat=-86.6, pvalue=0.0)`. On average ICU patients who are deceased have lower level of urine output compared to survived ICU patients.
- There is a statistically significant diffrence in blood potassium level between deceased and survived ICU patients `(tstat=-59.8, pvalue=0.0)`. On average we have higher number of ICU patients with low levels of blood potassium among deceased ICU patients.
- There is a statistically significant diffrence in blood sodium levels (sodium sapsii score=1 and 5) between deceased and survived ICU patients `(tstat=-64,-67, pvalue=0.0, 0.0)`. On average we have higher number of patients with low level of blood sodium among deceased ICU patients.
- There is a statistically significant diffrence in white blood cell counts between deceased and survived ICU patients `(tstat=-47, pvalue=0.0)`. On average we have higher proportion of ICU patients with above normal white blood cell counts among deceased ICU patients.
- There is no statistical diffrence in the proportion of patients with cardiac arrest (`hr_11.0`), however we observed statistically significant diffrence in the proportion of patients with extreme tachycardia (`hr_7.0, -79.4, 0.0`), and heart rate below normal (`hr_2.0, 137, 0.0).
- There is a statistically significant diffrence in the patient systolic blood pressure between deceased and survived ICU patients. On average we have higher number of ICU patients with systeolic blood pressure less than 70mmHg `(tstat=90.8, pvalue=0.0)`, below normal (between 70 -99mmHg), `(tstat=162, pvalue=0.0)`, above normal (more than and equal to 200mmHg) `(tstat=-70.7, pvalue=0.0)` among deceased ICU patients.

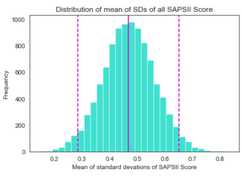
Inferential Statistics - Bootstrapping

Hypothesis testing based on bootstrapped survived and deceased dataseet at 5% significance level shows significant diffrence in mean SAPSII total score between bootstraped datasets of survived and deceased icu patients.

In [20]:
```
survived=saps.saps2.loc[saps.death==0]
deceased=saps.saps2.loc[saps.death==1]

np.random.seed(47)
bs_std_diff=np.empty(N_rep)

for i in range (N_rep):
    bs_survived = np.random.choice(survived, size=len(survived))
    bs_deceased = np.random.choice(deceased, size=len(deceased))
    bs_std_diff[i]=np.std(bs_deceased) - np.std(bs_survived)

bs_std_diff_mean, bs_std_diff_std=np.mean(bs_std_diff), np.std(bs_std_diff)
ci_std_diff=[bs_std_diff_mean - 1.96*bs_std_diff_std, bs_std_diff_mean + 1.96*bs_std_diff_std]

print('The 95% confidence interval for the difference between the standard deviations\
 of survived and deceased SAPSII Score is: ', ci_std_diff)
```

```
The 95% confidence interval for the difference between the standard deviations of survived and deceased SAPSII Score
is:  [0.28484245733704616, 0.6520856816890048]
```

In [21]:
```
# Plot the histogram of values and mark the locations of the percentiles
_ = plt.hist(bs_std_diff, bins=30, linewidth=0.5, color='turquoise')
_ = plt.xlabel('Mean of standard devations of SAPSII Score')
_ = plt.ylabel('Frequency')
_ = plt.title('Distribution of mean of SDs of all SAPSII Score')
_ = plt.axvline(bs_std_diff_mean, color='m')
_ = plt.axvline(ci_std_diff[0], color='m', linestyle='--')
_ = plt.axvline(ci_std_diff[1], color='m', linestyle='--')
```



The 95% confidence interval for the difference between the standard deviations of survived and deceased SAPSII Score is: [0.28484245733704616, 0.6520856816890048]

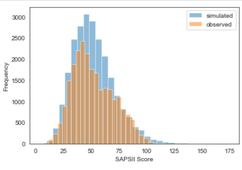
In [23]:
```
#Define a function to generate a permutation sample from insured and uninsured datasets
def permutation_sample(data1, data2):

    # Concatenate the data sets: data
    data = np.concatenate((data1, data2))

    # Permute the concatenated array: permuted_data
    permuted_data = np.random.permutation(data)

    # Split the permuted array into two: perm_sample_1, perm_sample_2
    perm_sample_1 = permuted_data[:len(data1)]
    perm_sample_2 = permuted_data[len(data1):]

    return perm_sample_1, perm_sample_2
```

In [24]:
```
#A bootstrap hypothesis test for difference of means
np.random.seed(47)
# Compute the difference in mean charges: diff_means
diff_means=np.mean(deceased) - np.mean(survived)

deceased_shifted=deceased - np.mean(deceased) + np.mean(survived)
perm_mean_replicates=np.empty(N_rep)
for i in range(N_rep):
    perm_survived, perm_deceased=permutation_sample(survived, deceased_shifted)
    perm_mean_replicates[i]=np.mean(perm_survived) - np.mean(perm_deceased)

bs_mean_diff=np.empty(N_rep)
for i in range(N_rep):
    bs_mean_diff[i]=np.mean(bs_survived) - np.mean(np.random.choice(deceased_shifted, len(deceased_shifted)))

# Compute p-value: perm_p, p
perm_p = np.sum(perm_mean_replicates>=diff_means)/len(perm_mean_replicates)
print('Permuation Pvalue: ', perm_p)
p=np.sum(bs_mean_diff>=diff_means)/len(bs_mean_diff)
print('Pvalue: ', p)
```

```
Permuation Pvalue:  0.0
Pvalue:  0.0
```

Inferential Statistics - Bayesian

In [29]:
```
_ = plt.hist(deceased_model_rvs, bins=30, alpha=0.5, label='simulated')
#Initial simulation -
seed(47)
n_survived = len(survived)
n_deceased = len(deceased)
deceased_model_rvs = gamma(alpha_est, scale=beta_est).rvs(n_deceased)
_ = plt.hist(deceased, bins=30, alpha=0.5, label='observed')
_ = plt.xlabel('SAPSII Score')
_ = plt.ylabel('Frequency')
_ = plt.legend()
```



In [44]:
```
with pm.Model() as a_model:
    alpha_ = pm.Exponential('alpha', 1/alpha_est)
    rate_ = pm.Exponential('beta', 1/rate_est)
    saps_deceased =pm.Gamma('Saps2_score_deceased', alpha=alpha_, beta=rate_, observed=deceased)
    trace = pm.sample(10000, tune=2000, cores=4)
```

```
Auto-assigning NUTS sampler...
Initializing NUTS using jitter+adapt_diag...
Multiprocess sampling (4 chains in 4 jobs)
NUTS: [beta, alpha]
Sampling 4 chains: 100%|[redacted]| 48000/48000 [01:22<00:00, 580.59draws/s]
The number of effective samples is smaller than 25% for some parameters.
```

In [45]:
```
alpha_samples = trace['alpha']
beta_samples = trace['beta']
alpha_ci = np.percentile(alpha_samples, [2.5, 97.5])
beta_ci = np.percentile(beta_samples, [2.5, 97.5])
print('95% confidence interval for alpha: ', alpha_ci)
print('95% confidence interval for beta: ', beta_ci)
```

```
95% confidence interval for alpha:  [7.33467864 7.59837212]
95% confidence interval for beta:  [0.14113107 0.14638464]
```

Clinical Deterioration Prediction - Machine Learning Models

Machine Learning Model - Logistic Regression

This notebook contains code and a clear record of all steps in developing a predictive variable and dataset for a CU Boulder - Hospital Medicine pilot to target patients.

```
In [1]: # Common packages for features and target variable
        import numpy as np
        import pandas as pd
        import matplotlib.pyplot as plt

        # Machine learning packages
        from sklearn.model_selection import train_test_split
        from sklearn.preprocessing import StandardScaler, LabelEncoder
        from sklearn.metrics import confusion_matrix, classification_report

        # Keras packages
        from keras.models import Sequential
        from keras.layers import Dense

        # Print the dimensions of X and y after cleaning
        print("Dimensions of y after cleaning: {0}".format(y.shape))
        print("Dimensions of X after cleaning: {0}".format(X.shape))

        Dimensions of y after cleaning: (20147,)
        Dimensions of X after cleaning: (20147, 7)
        Dimensions of y after cleaning: (20147,)
        Dimensions of X after cleaning: (20147, 7)

        ~ Split the data into a training and test (holdout) set
        ~ Train on the training set, and test for accuracy on the testing set

In [2]: from sklearn.model_selection import train_test_split
        from sklearn.linear_model import LogisticRegression
        from sklearn.metrics import accuracy_score, f1_score

        # Split the data into a training and test set
        X_train, X_test, y_train, y_test = train_test_split(X, y, random_state=1)

        # Initialize a logistic regression model
        lr = LogisticRegression(fit_intercept=True)

        # Train the classifier on the training data
        lr.fit(X_train, y_train)

        # Print the accuracy
        print("Training Accuracy: {:".format(lr.score(X_train, y_train)))
        print("Testing Accuracy: {:".format(lr.score(X_test, y_test)))

        Training Accuracy: 0.7077777777777777
        Testing Accuracy: 0.6956521739130435
```

Hyperparameter Tuning

The model has some hyperparameters we can tune for hopefully better performance. In Logistic Regression, the most important parameter to tune is the regularization parameter, C. Note that the regularization parameter is not always part of the logistic regression model. The regularization parameter is used to control for unwanted bias in the statistical inference, and in many cases can be and often does it speak to a method of feature selection. We may want to tune this as we iterate over modeling.

```
In [3]: # Import the GridSearchCV and numpy
        from sklearn.model_selection import GridSearchCV
        import numpy as np

        # Set up the hyperparameter grid to search over
        param_grid = {'C': [0.001, 0.01, 0.1, 1, 10, 100]}

        # Initialize GridSearch using the logistic regression model, the parameter grid, and 5 folds
        gs_lr = GridSearchCV(lr, param_grid, cv=5)

        # Fit the GridSearch model to the training data.
        gs_lr.fit(X_train, y_train)

        # Find the best hyperparameter value for C.
        print("Best C: {}".format(gs_lr.best_params_['C']))

        Best C: 0.1

In [4]: # Print the mean cross-validation score for the best model.
        print("Best score for training data: {:".format(gs_lr.best_score_))

        Best score for training data: 0.7077777777777777

        # Refit the logistic regression model with the best value for C.
        lr = LogisticRegression(fit_intercept=True, C=0.1)

        # Fit the model.
        lr.fit(X_train, y_train)

        # Print the accuracy with the test data.
        print("The accuracy with the test data is {:".format(lr.score(X_test, y_test)))

        The accuracy with the test data is 0.6956521739130435
```

Running the model with C=0.1 gives us the same accuracy results on the test data as the default. This is not always the case before improved to experiment with the hyperparameters that some tune with new data.

Grid Search

```
In [5]: from sklearn.model_selection import GridSearchCV

        # Import numpy and the metrics from sklearn.
        import numpy as np
        from sklearn.metrics import f1_score, roc_auc_score, accuracy_score

        # Define the parameter grid to search over.
        param_grid = {'C': [0.001, 0.01, 0.1, 1, 10, 100],
                      'penalty': ['l1', 'l2'],
                      'fit_intercept': [True, False],
                      'solver': ['liblinear'],
                      'max_iter': [100, 200, 300]}

        # Initialize GridSearch using the logistic regression model, the parameter grid, 5 folds, 1 core, scoring='accuracy', verbose=verbose=3.
        gs = GridSearchCV(lr, param_grid, cv=5, n_jobs=1, scoring='accuracy', verbose=3)

        # Fit the GridSearch model to the training data.
        gs.fit(X_train, y_train)

        Fitting 5 folds for each of 144 candidates, totalling 720 fits

        [CV] C=0.001, fit_intercept=True, penalty=l1, solver=liblinear, max_iter=100
        [CV] C=0.001, fit_intercept=True, penalty=l1, solver=liblinear, max_iter=100, total=  0.1s
```

Machine Learning Model - KNN

In [11]:

```
from sklearn.neighbors import KNeighborsClassifier
knn=KNeighborsClassifier(n_neighbors=6)
knn.fit(X,y)
```

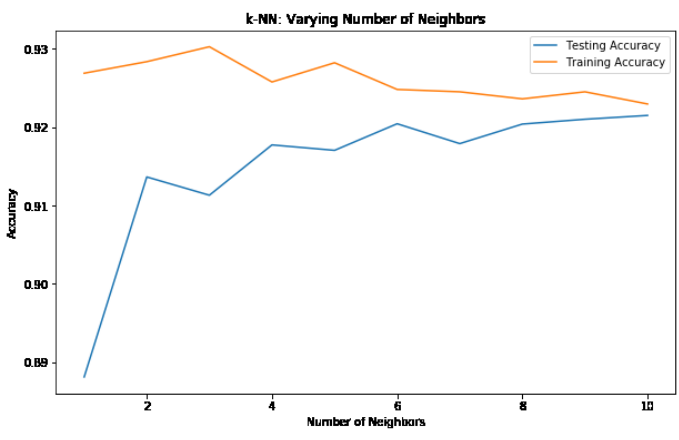
Out[11]:

```
KNeighborsClassifier(algorithm='auto', leaf_size=30, metric='minkowski',
                     metric_params=None, n_jobs=None, n_neighbors=6, p=2,
                     weights='uniform')
```

In [13]:

```
# Import necessary modules
from sklearn.neighbors import KNeighborsClassifier
from sklearn.model_selection import train_test_split


# Split into training and test set
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size = 0.3, random_state=42, stratify=y)

# Create a k-NN classifier with 10 neighbors: knn
knn = KNeighborsClassifier(n_neighbors=10)

# Fit the classifier to the training data
knn.fit(X_train, y_train)

# Print the accuracy
print('Training Accuracy: {}'.format(knn.score(X_train,y_train)))
print('Testing Accuracy: {}'.format(knn.score(X_test, y_test)))
```

```
Training Accuracy: 0.924405694116547
Testing Accuracy: 0.9198298429319371
```



In [15]:

```
# Import necessary modules
from sklearn.metrics import classification_report
from sklearn.metrics import confusion_matrix

# Create training and test set
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size =0.4, random_state=42)

# Instantiate a k-NN classifier: knn
knn = KNeighborsClassifier (n_neighbors=7)

# Fit the classifier to the training data
knn.fit(X_train, y_train)

# Predict the labels of the test data: y_pred
y_pred = knn.predict(X_test)

# Generate the confusion matrix and classification performance report
print(confusion_matrix(y_test, y_pred))
print(classification_report(y_test, y_pred))
```

```
[[21404   421]
 [ 1586  1036]]
              precision    recall  f1-score   support

           0       0.93      0.98      0.96     21825
           1       0.71      0.40      0.51      2622

    accuracy                           0.92     24447
   macro avg       0.82      0.69      0.73     24447
weighted avg       0.91      0.92      0.91     24447
```

In the support column - the firgures represent the number of ICU patients in the test set on which 21,825 are ICU stay survivors and 2,622 are non-survivors (deceased).

Our model shows 92% accuracy - represents the number of correctly classified (True positive and true negative) over the total number of data instances (true positive, false positive, true negative, and false negative).

Precision (positive predicitve value) in classifying the data instances. Defined as true positive over true positive plus false positive. Our model have a 93% precision in classifying ICU stay survivors as survivors and a 71% precision classifying ICU patient deaths correctly.

Recall (sensetivity or ture positive rate). Defined as true positive over true positive plus false negative. 100% recall means there is zero false negative. Our model have a high (98%) recall classifying ICU stay survivors (there only 2% false negative - survivors clasfied as deceased. Model sensetivity classifying deceased is only 40%, that is 60% fase negative - deceased classified as survivors.

so, ideally in a good classifier, we want a metric that takes into account both precision and recall. We have f1-score for that. f1-score becomes high only if both precision and recall becomes high. Our model shows f1-score of 96% for classifying survivors and 51% classifying deceased.

Machine Learning Model - Ensemble Algorithms

In [13]:
```
# Evaluate several ml models by training on training set and testing on testing set
def evaluate(X_train, X_test, y_train, y_test):
    #Names of models
    model_name_list = ['Random Forest', 'Extra Trees',
                       'Gradient Boosted']
    model = np.arange(1, 4)
    train_accuracy = np.empty(len(model))
    test_accuracy = np.empty(len(model))
    # Instantiate the models
    model1 = RandomForestClassifier(n_estimators=100)
    model2 = ExtraTreesClassifier(n_estimators=100)
    model3 = GradientBoostingClassifier(n_estimators=100)
    results =pd.DataFrame(columns=['train_accuracy', 'test_accuracy'])

    # Train and predict with each model
    for i, model in enumerate([model1, model2, model3]):
        model.fit(X_train, y_train)

        train_accuracy=model.score(X_train, y_train)
        test_accuracy=model.score(X_test, y_test)

        model_name=model_name_list[i]
        results.loc[model_name, :] =[train_accuracy, test_accuracy]
    return results
```

In [14]:
```
results = evaluate(X_train, X_test, y_train, y_test)
results
```

Out[14]:

	train_accuracy	test_accuracy
Random Forest	0.949323	0.91983
Extra Trees	0.949323	0.919339
Gradient Boosted	0.924149	0.925993

All selected ensemble prediction algorithms out performed the `logistic regression` model. `Random forest` and `Extra trees` algorithm achieved training data accuracy of 95% in discriminating between survived and deceased patients in the training data and 92% accuracy in discriminating test data.

In [ ]: