A4 Feature Engineering [⁘]

Welcome to A4!

Please enter answers to the questions in the specified Markdown cells below, and complete the code snippets in the associated python files as specified. When you are done with the assignment, follow the instructions at the end of this assignment to submit.

Learning Objective 🌱

In this assignment, you will gain experience transforming clinical data into sets of features for downstream statistical analysis, utilizing the cohort that you developed in A3. In particular, you will extract features from vitals, diagnosis codes, and more that can be used to predict the future development of septic shock. You will practice using common time-saving tools in the **Pandas** 🐼 library and **Python** 🐍 programming language that are ideally suited to these tasks.

Resources 📖

- Pandas Cheat Sheet 🐼 : https://pandas.pydata.org/Pandas_Cheat_Sheet.pdf
- Relevant publications:
    - You will not be replicating the models presented in "A targeted real-time early warning score (TREWScore) for septic shock" by Henry et al. directly, but we include a link to the paper for your reference.

Environment Set-Up 🐍

To begin, we will need to set up an virtual environment with the necessary packages. A virtual environment is a self-contained directory that contains a Python interpreter (aka Python installation) and any additional packages/modules that are required for a specific project. It allows you to isolate your project's dependencies from other projects that may have different versions or requirements of the same packages.

In this course, we require that you utilize Miniconda to manage your virtual environments. Miniconda is a lightweight version of Anaconda, a popular Python distribution that comes with many of the packages that are commonly used in data science.

Instructions for setting up your environment using Miniconda:

1. If you do not already have Miniconda installed, download and install the latest version for your opperating system from the following link: https://docs.conda.io/en/latest/miniconda.html#latest-miniconda-installer-links

2. Create a new virtual environment for this assignment by running the following command in your terminal:

   `conda env create -f environment.yml`
   This will create a new virtual environment called `biomedin215`

3. Activate your new virtual environment by running the following command in your terminal:

   `conda activate biomedin215`
   This will activate the virtual environment you created in the previous step.

4. Finally, ensure that your `ipynb` (this notebook)'s kernel is set to utilize

the `biomedin215` virtual environment you created in the previous steps. Depending on which IDE you are using to run this notebook, the steps to do this may vary.

In [ ]:
```
# Run this cell:
# The lines below will instruct jupyter to reload imported modules before
# executing code cells. This enables you to quickly iterate and test revisic
# to your code without having to restart the kernel and reload all of your
# modules each time you make a code change in a separate python file.

%load_ext autoreload
%autoreload 2
```

In [ ]:
```
# Run this cell to ensure the environment is setup properly
# If you get an error, please ensure that the environment was activated for
# Note: You do not need to edit this cell
import pandas as pd
import os
import warnings

print("Imports Successful!")
```

```
Imports Successful!
```

Note to Students: 📚

> Throughout the assignment, we have provided `sanity checks`: small warnings that will alert you when your implementation is different from the solution. Our goal in providing these numbers is to help you find bugs or errors in your code that may otherwise have gone unnoticed. Please note: the sanity checks are just tools we provided to be helpful, and should not be treated as a target to hit. We manually grade each assignment based on

> the code you submit, and not based on whether you get the exact same numbers as the sanity checks.

Even if you are failing the sanity checks, if your implementation is correct with minor errors, you will still receive the majority of the points (if not all).

In [ ]:
```
# Run this cell to set up sanity checks warnings
# Note: You do not need to change anything in this cell

# Creates a custom warning class for sanity checks
class SanityCheck(Warning):
    pass

# Sets up a cosutom warning formatter
def custom_format_warning(message, category, filename, lineno, line=None):
    if category == SanityCheck:
        # Creates a custom warning with orange text
        return f'\033[38;5;208mSanity Check - Difference Flagged:\n{message}

    return '{}:{}: {}: {}\n'.format(filename, lineno, category.__name__, mes

# Sets the warning formatter for the entire notebook
warnings.formatwarning = custom_format_warning
```

Data Description 📂

We will be utilizing the same subset of the MIMIC III database we utilized in A3: the 1,000 subject development cohort you created previously. You will start with a dataset very similar to what you may have generated at the end of the prior assignment.

You will analyze the available data to identify a cohort of patients that underwent septic shock during their admission to the ICU. **All of the data you need for this assignment is available on Canvas.**

Once you have downloaded and unzipped the data, you should see the following `7` csv files:

- `cohort_labels.csv`
- `ADMISSIONS.csv`
- `DIAGNOSES_ICD.csv`
- `notes_small_cohort_v2.csv`
- `snomed_ct_isaclosure.csv`
- `snomed_ct_str_cui.csv`
- `vitals_small_cohort.csv`

Specify the location of the folder containing the data in the following cells:

In [ ]:
```
# Specify the path to the folder containing the data files
data_dir = "./data" # <-- TODO: You will need to change this path
```

In [ ]:
```
# Run this cell to make sure all of the files are in the specified folder
expected_file_list = ["cohort_labels.csv", "ADMISSIONS.csv", "DIAGNOSES_ICD.

for file in expected_file_list:
    assert os.path.exists(os.path.join(data_dir, file)), "Can't find file {}

print("All files successfully found")
```

```
All files successfully found
```

1. Defining labels for prediction 🟩🟩🟩🟥🟩🟩🟥🟩

`1.1:(10 pts)`

Utilizing our version of the 1,000 subject development cohort you created in the previous assignment, in this assignment, your task is to engineer a set of features that will be used as the inputs to a model that will predict:

> At 12 hours into an admission, whether septic shock will occur during the remainder of the admission, with at least 3 hours of lead time (the amount of time between when an event is predicted to occur and when it actually occurs).

To begin, let's load in our intial dataframes.

- `cohort_labels.csv`: contains the cohort with the various labels we defined in A3. (This dataset will probably look very similar to the dataset you had at the end of A3.)
- `ADMISSIONS.csv`: an extract of the ADMISSIONS table from MIMIC-III. This contains information about patient admission events to the hospital.

In [ ]:
```
# Run this cell to load the data from the CSV files into Pandas DataFrames
# Note: You do not need to change anything in this cell

# Reads in the tables from the CSV files
cohort_labels = pd.read_csv(os.path.join(data_dir, "cohort_labels.csv"))
admissions = pd.read_csv(os.path.join(data_dir, "ADMISSIONS.csv"))

# Sets the column names to be lowercase
admissions.columns = [x.lower() for x in admissions.columns]
```

In [ ]:
```
# Run this cell to view what the first few rows of the cohort_labels table l
# Note: You do not need to change anything in this cell
cohort_labels.head(3)
```

Out[ ]:

	subject_id	hadm_id	icustay_id	charttime	sepsis	severe_sepsis	septic_shock
0	3	145834	211552.0	2101-10-20T16:40:00Z	False	False	False
1	3	145834	211552.0	2101-10-20T16:49:00Z	False	False	False
2	3	145834	211552.0	2101-10-20T19:12:00Z	False	False	False

In [ ]:
```
# Run this cell to view what the first few rows of the admissions table look
admissions.head(3)
```

Out[ ]:

	row_id	subject_id	hadm_id	admittime	dischtime	deathtime	admission_type	admissio
0	21	22	165315	2196-04-09 12:26:00	2196-04-10 15:54:00	NaN	EMERGENCY	EMERGE
1	22	23	152223	2153-09-03 07:15:00	2153-09-08 19:10:00	NaN	ELECTIVE	REFERRA
2	23	23	124321	2157-10-18 19:34:00	2157-10-25 14:00:00	NaN	EMERGENCY	TRANS HOS

In [ ]:
```
# (OPTIONAL TODO:) It is always a good idea to filter out columns that you d
# As always, feel free to add code to your notebooks to do this. This is not
# You may want to come back to this later when you are more familiar with th
```

First, we need to do some preprocessing. When working with dates in Pandas, it is always a good idea to convert the data to a datetime format. This can help improve performance, memory efficiency, and also allow us to use the many built-in features of Pandas that are only available for datetime objects. Implement the function `preprocess_dates` in the file `src/utils.py` following the instructions in the docstring, to convert specific columns in the input dataframe that contain dates to datetime objects.

In [ ]:
```
# Run this cell after you have completed the necessary code
# Note: you do not need to modify the code in this cell
from src.utils import preprocess_dates

preprocess_dates(admissions, ["admittime", "dischtime"], ["%Y-%m-%d %H:%M:%S
preprocess_dates(cohort_labels, ["charttime"], ["%Y-%m-%dT%H:%M:%SZ"],inplac

#==============
# Sanity Checks
if not "datetime" in str(admissions["admittime"].dtype).lower():
```

```
    warnings.warn("The admittime column is not a datetime object", SanityChe

if not admissions["admittime"].dt.tz is not None:
    warnings.warn("The admittime column is not in UTC", SanityCheck)
```

Next, we will derive the **labels** and **index times** in a way that aligns with the task description above. Note that we are no longer following the same procedure as the TREWScore paper.

We will use the following definitions:

- We will only assign labels to admissions of at least twelve hours in duration.
- An admission is assigned a negative label if septic shock does not occur at any time during the admission.
- An admission is assigned a positive label if septic shock occurs fifteen hours after admission or later.
- Admissions where the earliest time of septic shock occurs prior to fifteen hours after admission are removed from the study.
- For admissions that have valid labels, we assign an index time at twelve hours into the admission. For prediction, we only use information that occurs before the index time.
- In the case that a patient has multiple admissions for which a valid index time and label may be assigned, we only use the latest one.

We will use the above definitions to derive the binary classification labels for septic shock and the corresponding index times for each patient in the dataframe. Our goal is to end up with a dataframe that contains a row for each patient in `cohort_labels` that passed the inclusion criteria, with the following columns:

- `subject_id`: the unique identifier for each patient
- `hadm_id`: the unique identifier for the admission
- `label`: the binary classification label for septic shock
- `index_time`: the index time for the patient (+12 hours from admission start time)

As mentioned above, we do not want to assign labels to admissions that are less than twelve hours in duration. Implement the function `filter_admissions` in `src/labels.py` following the instructions in the docstring and run the following cell.

In [ ]:
```
# Run this cell after you have completed the necessary code
# Note: you do not need to modify the code in this cell
from src.labels import filter_admissions

filtered_admissions = filter_admissions(admissions)
```

```
#==============
# Sanity Checks

# Check that the number of rows is correct
if filtered_admissions.shape[0] != 57925:
    warnings.warn("Number of rows is different than expected", SanityCheck)
```

Our next step will be to merge the two dataframes together, and create two additional columns:

- `relative_charttime`: The amount of time between the charttime and the start of the admission
- `index_time`: The time at which a prediction will be made (12 hours after the start of the admission)

Implement the functions `merge_and_create_times`, `get_relative_charttime`, and `get_index_time` in `src/labels.py` following the instructions in the appropriate docstrings. When you are done, run the cell below to sanity check your implementation.

In [ ]:
```
# Run this cell after you have completed the necessary code
# Note: you do not need to modify the code in this cell
from src.labels import merge_and_create_times

merged_cohort = merge_and_create_times(cohort_labels,filtered_admissions)

#==============
# Sanity Checks
if merged_cohort.shape[0] != 247610:
    warnings.warn(f"Number of rows is different than expected: shape[0] = {m
```

Now we need to use this merged dataframe to create a new dataframe that contains the labels utilizing the definitions above. Implement the function `get_shock_labels` in `A4/labels.py` following the instructions in the docstring to create a new dataframe with a binary septic shock label for each patient.

In [ ]:
```
# Run this cell after you have completed the necessary code
# Note: you do not need to modify the code in this cell
from src.labels import get_shock_labels

# test1,test2 = get_shock_labels(merged_cohort)

shock_labels = get_shock_labels(merged_cohort)


#==============
# Sanity Check:
for col in shock_labels.columns:
    if col not in ["subject_id", "hadm_id", "admittime", "dischtime", "index
        warnings.warn(f"Expected column {col} not found", SanityCheck)
```

```
if len(shock_labels) != 974:
    warnings.warn(f"Expected length different: length = {len(shock_labels)}"

if len(shock_labels) != shock_labels["subject_id"].nunique():
    warnings.warn(f"Expected no duplicate rows", SanityCheck)
```

In [ ]:
```
# Run this cell to see the class balance of the labels:
# Note: you do not need to modify the code in this cell
shock_labels["label"].value_counts()
```

Out[ ]:
```
label
False    905
True      69
Name: count, dtype: int64
```

2 Feature engineering [⣿]

Now that we have derived labels and index times for each patient in our cohort, we can start to engineer some features from the data that occur prior to the index times and will be useful for predicting onset of septic shock.

First lets deal with diagnoses. Load in the `DIAGNOSES_ICD.csv` file by running the cell below.

In [ ]:
```
# Run this cell to load the data from the CSV files into Pandas DataFrames
# Note: You do not need to modify the code in this cell

# Reads in the table from the CSV file
diagnoses = pd.read_csv(os.path.join(data_dir, "DIAGNOSES_ICD.csv"))
# Sets diagnoses's column names to lower case
diagnoses.columns = [x.lower() for x in diagnoses.columns]
```

`2.1:(2 pts)`

Review the documentation for MIMIC to answer the following question.

Which column from which table in MIMIC should you use to find the time of each diagnosis? Justify your response.

According to MIMIC-III documentations, final diagnoses for a patient's hospital stay are coded on discharge and can be found in the DIAGNOSES_ICD table (https://mimic.mit.edu/docs/iii/tables/admissions/). This means that the column DISCHTIME from ADMISSION table can be used to determine time of diagnosis.

`2.2:(3 pts)`

Utilizing the column you selected in the previous question, implement the function `get_diagnoses` in `A4/features.py` following the instructions in the docstring. When you have completed your implementation, run the cell below to sanity check.

In [ ]:
```
from src.features import get_diagnoses

dx_features = get_diagnoses(admissions, diagnoses, shock_labels)

#============
# Sanity Check
if dx_features.shape[0] != 4031:
    warnings.warn(f"Expected length different: shape[0] = {dx_features.shape
```

How many subjects have diagnoses recorded prior to the index_time? Does the resulting number make sense?

In [ ]:
```
# TODO: Add code to this cell to answer the above question if needed
dx_features['subject_id'].unique().shape[0]
```

Out[ ]:
```
210
```

A total of 210 unique sujects have diagnosis prior to index_time. This makes sense as that means 210 patients have had a diagnosis made within 12 hours in one of his/her admissions. This is expected to be larger than the number of patients with True shock_label from part 1, since that only accounts for the lastest admission to label the patients.

2.3:(4 pts)

Implement code in the following cell to answer the question

What are the top 10 most common diagnosis codes (by number of unique patients who had the code in their history) in the data frame resulting from question 2.2? Look up the top 3 codes online and report what they refer to.

In [ ]:
```
# TODO: IMPLEMENT CODE HERE TO ANSWER THIS QUESTION

unique_subject_ids = dx_features.groupby(by='icd9_code', as_index=False)['su

unique_subject_ids['count'] = [len(x) for x in unique_subject_ids['subject_i
```

In [ ]:
```
unique_subject_ids.sort_values('count', ascending=False)[:3]
```

Out[ ]:

	icd9_code	subject_id	count
255	4019	[23, 36, 357, 362, 94, 103, 124, 154, 156, 117...	93
308	4280	[34, 357, 68, 107, 130, 138, 156, 117, 21, 305...	77
273	41401	[23, 34, 36, 357, 85, 107, 124, 130, 138, 154,...	73

In [ ]:
```
## top 10 codes:
unique_subject_ids.sort_values('count', ascending=False)[:10].icd9_code.valu
```

Out[ ]:
```
array(['4019', '4280', '41401', '5849', '42731', '2859', '5990', '25000',
       '51881', '486'], dtype=object)
```

Top 3 ICD9 codes are 4019, 4280, and 41401. They corresponds to:

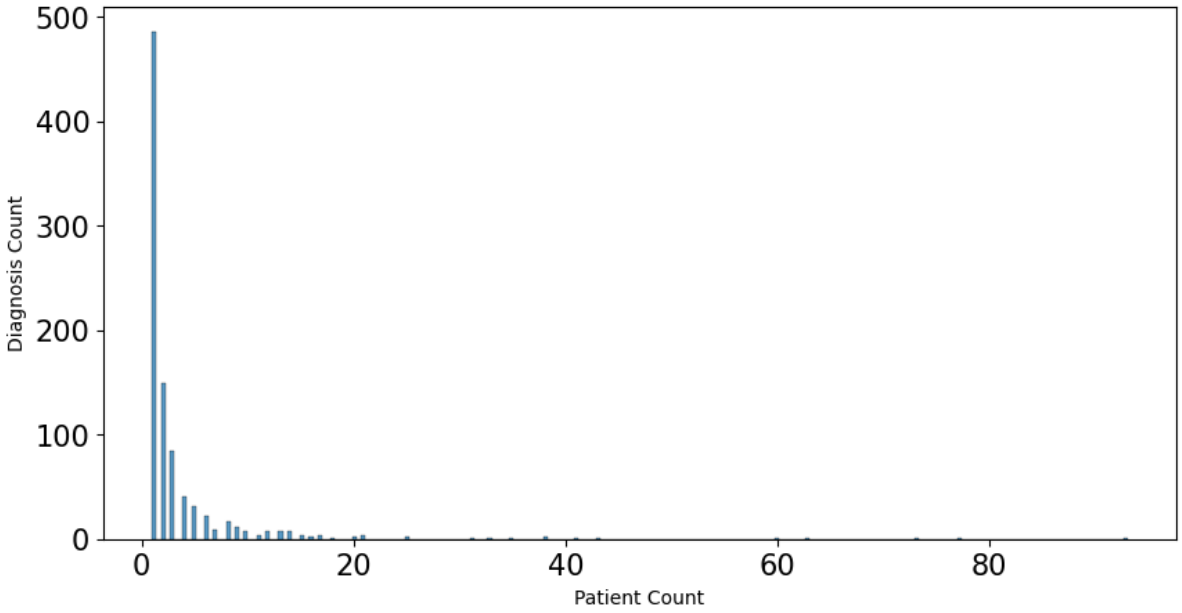
- 4019: Hypertension NOS, Unspecified essential hypertension
- 4280: CHF NOS, Congestive heart failure, unspecified
- 41401: Coronary atherosclerosis of native coronary artery

2.4:(4 pts)

In this step we will create a histogram for the set of codes and patients that remain after the index time filtering step.

Implement the function `show_diagnosis_hist` in `visualize.py` following the instructions in the docstring. When you are done, run the cell below to show the histogram.

In [ ]:
```
from src.visualize import show_diagnosis_hist

# Create the plot
show_diagnosis_hist(dx_features, "diagnosis_count_hist.png")
```



In 1-2 sentences, interpret the resulting histogram.

There is a high diversity of diagnosis among patients, as there are almost 500 diagnosis made to unique patients (highest bin at 1 diagnosis and under 10 patient count). There are very small number of diagnosis assigned to large amount of patients in this cohort (indicated by the bins at patient count > 60).

`2.5:(5 pts)`

From the histogram you generated earlier, it's evident that there's a substantial variation in the frequency of different diagnoses. Specifically, a significant number of diagnoses appear very infrequently in the dataset.

Such a distribution is characteristic of a `sparse feature space`. Here is what that means:

`Sparse Feature Space`: In the context of data with categorical variables, a sparse feature space refers to the scenario where many possible features (in this case, diagnosis codes) appear infrequently, resulting in a 'wide form' matrix with many zeros or absent values.

This can have problematic implications for downstream analyses:

- First, sparse features can pose **computational challenges**: Many machine learning algorithms struggle with high dimensionality and sparsity. They can become computationally intensive or may not work optimally.
- Second, sparse features can lead to **issues with generalization**: Rare features often don't contribute significantly to model training. In some cases, they might

even introduce noise, making the model overfit to a training set and perform poorly on new, unseen data.

Given these challenges, it's beneficial to address sparsity. One strategy to manage this involves quantifying the "usefulness" or "specificity" of each feature, and utilizing this information to select features or even perform feature aggregation (grouping features to capture broader patterns). This is where `Information Content (IC)` comes into play:

Definition: `IC` is a metric that provides a measure of the specificity or the informativeness of a feature based on its frequency of occurrence. Features that are very common have a higher probability and thus a lower IC, while rare features have a lower probability, resulting in a high IC value.

The IC of a feature that occurs in a set of records is calculated as follows:

$$IC\left(\text{featureA}\right) = -log_2\left(\frac{count(\text{Patients with featureA})}{count(\text{All Patients})}\right)$$

Implement the function `calc_ic` in `src/features.py` to calculate the IC of each diagnosis code in the dx_features dataframe using the equation above and following the instructions in the docstring. When you are done, run the cell below to sanity check your implementation.

In [ ]:
```
from src.features import calc_ic

icd9_ic = calc_ic(dx_features, all_patients_count=len(shock_labels))

if icd9_ic.shape[0] != 914:
    warnings.warn(f"Expected number of rows different: shape[0] = {icd9_ic.s
```

`2.6` (`3 pts`)

Use the code cell below to answer the following question:

What is the range (min and max) of ICs observed in your data? What are the 10 most specific ICD9 codes?

In [ ]:
```
# TODO: IMPLEMENT CODE HERE TO ANSWER THE QUESTION

icd9_ic['icd9_ic'].max(), icd9_ic['icd9_ic'].min(), icd9_ic.sort_values('icc
```

Out[ ]:
```
(9.927777962082342,
 3.3886191509743107,
 array(['V1007', '4580', '4263', '7210', '37852', '5307', '9828', 'E9509',
        '5680', '29620'], dtype=object))
```

The min IC is 3.388 while the max IC is 9.92777.

According to definition for IC, the features with higher specificity/informativeness are rare and have high IC, hence top 10 codes are: 'V1007', '4580', '4263', '7210', '37852', '5307', '9828', 'E9509', '5680', '29620'

2.7 (2 pts)

Now it's time to perform some feature selection. Implement the function `filter_ic` in `src/features.py` to filter the dataframe to only include the diagnoses with an IC between 4 and 9 (inclusive) following the instructions in the docstring. When you are done, run the cell below to sanity check your implementation.

In [ ]:
```
from src.features import filter_ic

dx_selected = filter_ic(dx_features, icd9_ic)

#============
# Sanity Check
if dx_selected.shape[0] != 3044:
    warnings.warn(f"Expected number of rows different: shape[0] = {dx_select
```

2.8 (12 pts)

Now we have our diagnosis features and the times they occurred for each patient. The next step is to create a patient-feature matrix that summarizes and organizes these diagnosis features. In this matrix, each row should represent a patient and each column should represent a diagnosis code, time-binned by whether or not it occurred in the 6 months prior to the index time.

Put simply, for each diagnosis code, we want to generate two features:

- One feature representing the count of the number of times the code was observed in the six months prior to the index time.
- Another feature for the number of times that code appeared more than six months before the index time.

Note that the ICU stay is the first time many patients have been seen at this hospital, so patients may have few or no prior recorded diagnoses.

Implement the function `get_diagnoses_features` in `src/features.py` to create the patient-feature matrix following the instructions in the docstring. When you are done, run the cell below to sanity check your implementation.

In [ ]:
```
from src.features import get_diagnosis_features

diagnosis_features = get_diagnosis_features(dx_selected)
```

```
#============
# Sanity Check

if diagnosis_features.shape[0] != 209:
    warnings.warn(f"Expected number of rows different: shape[0] = {diagnosis
```

2.9 (4 pts)

Now let's add features from notes. To do so, we'll have to process some text.

The `noteevents` table in MIMIC is large and unwieldy, so we've extracted the rows from that table that you will need. The result is in the file `notes_small_cohort_v2.csv`. Let's load this in now.

In [ ]:
```
# Run this cell to load the data from the CSV files into Pandas DataFrames
# Note: You do not need to modify the code in this cell

# Reads in the table from the CSV file
notes = pd.read_csv(os.path.join(data_dir, "notes_small_cohort_v2.csv"))

# Set notes' column names to lower case
notes.columns = [x.lower() for x in notes.columns]

# Utilizes the preprocess_dates function to convert the dates to datetime ob
preprocess_dates(notes, ["chartdate"], ["%Y-%m-%d"], inplace=True)
```

In [ ]:
```
# Let's check out what the notes data looks like
notes.head(3)
```

Out[ ]:

	row_id	subject_id	hadm_id	chartdate	charttime	storetime	category	descriptio
0	44005	3	145834	2101-10-31 00:00:00+00:00	NaN	NaN	Discharge summary	Repor
1	94503	3	145834	2101-10-21 00:00:00+00:00	NaN	NaN	Echo	Repor
2	94502	3	145834	2101-10-21 00:00:00+00:00	NaN	NaN	Echo	Repor

In the MIMIC database, notes are primarily timestamped using the chartdate column, which captures the date (but not the precise time) when the note was recorded. Another column, charttime, exists, but it is predominantly empty or null for most entries. This presents a challenge when we wish to filter notes based on precise times, such as a patient-specific cutoff time.

To address this, our approach will be to filter notes by ensuring that they were recorded strictly before the day corresponding to each patient's `index_time`. This means that if

a note's chartdate is the same as the `index_time` (*even if charttime were available*), we would exclude it because we can't ascertain if it was before or after the exact `index_time` time on that day.

Implement the function `filter_by_chartdate` in `src/notes.py` to filter the notes dataframe to only include notes in a patient's record that were recorded before the day corresponding to each patient's `index_time`, following the instructions in the docstring. When you are done, run the cell below to sanity check your implementation.

In [ ]:
```
from src.notes import filter_by_chartdate

notes_filtered = filter_by_chartdate(shock_labels, notes)

#============
# Sanity Check
if notes_filtered.shape[0] != 13213:
    warnings.warn(f"Number of rows differs from expected: shape[0] = {notes_
```

2.10 (2 pts)

The Unified Medical Language System (UMLS) is a multi-dimensional and dynamic compendium developed by the U.S. National Library of Medicine (NLM) to bridge the gap between various healthcare terminologies and classification systems. At the heart of UMLS lie various terminologies, which provide concept hierarchies as well as sets of terms for individual concepts. For example, there are more than 50 terms in UMLS terminologies for the concept `myocardial infarction`!

Here we will use the SNOMED CT (Systematized Nomenclature of Medicine - Clinical Terms): A comprehensive clinical terminology encompassing diseases, clinical findings, procedures, etc. SNOMED CT is a multi-hierarchy system, meaning that each concept can have multiple parents. For example, the concept `myocardial infarction` has two parents: `acute coronary syndrome` and `myocardial disorder`.

In this assignment, you will use the SNOMED CT hierarchy and UMLS term sets to construct a dictionary of terms for inflammatory disorders, which you will use to search for associated terms in MIMIC III notes to create additional features.

First, load `snomed_ct_isaclosure.csv` and `snomed_ct_str_cui.csv` by running the code in the following cell:

In [ ]:
```
# Run this cell to load the data from the CSV files into Pandas DataFrames
# Note: You do not need to modify the code in this cell

# Reads in tables from the CSV files
snomed_ct_isaclosure = pd.read_csv(os.path.join(data_dir, "snomed_ct_isaclos
snomed_ct_str_cui = pd.read_csv(os.path.join(data_dir, "snomed_ct_str_cui.cs
```

In [ ]:
```
# snomed_ct_isaclosure contains the child-parent CUI relationships for all c
# Note: You do not need to modify the code in this cell
snomed_ct_isaclosure.head(3)
```

Out[ ]:

	descendant	ancestor	dist
0	C0038891	C0038891	0
1	C0038891	C0220806	3
2	C0038891	C0033684	2

In [ ]:
```
# snomed_ct_str_cui contains the terms (each with a unique term identifier,
# Note: You do not need to modify the code in this cell
snomed_ct_str_cui.head(3)
```

Out[ ]:

	tid	str	CUI
0	1	and	C1706368
1	2	0	C0919414
2	3	of	C0332285

Implement the function `merge_snomed` in `src/notes.py` to merge the two dataframes together, following the instructions in the docstring. When you are done, run the cell below to sanity check your implementation.

In [ ]:
```
from src.notes import merge_snomed

# Merge the two tables
snomed_ct_concept_str = merge_snomed(snomed_ct_isaclosure, snomed_ct_str_cui

#============
# Sanity Check
if snomed_ct_concept_str.shape[0] != 16407662:
    warnings.warn(f"Number of rows differs from expected: shape[0] = {snomed

if snomed_ct_concept_str.shape[1] != 2:
    warnings.warn(f"Number of columns differs from expected: shape[1] = {snc
```

2.11 (6 pts)

One feature that is very likely to impact the likelihood of a patient to develop septic shock is whether they currently have or have a history of inflammatory disorders. Let's extract information from clinical notes to look for the presence of this class of disease.

To accomplish this, implement the function `get_cui_list` in `src/notes.py` to get a list of all the terms that correspond to a CUI in the `snomed_ct_isaclosure` dataframe and that have a specified number of characters or fewer, following the instructions in the docstring. Then, use this function to get a set of terms for

`inflammatory disorders` (`C1290884`) that have 20 characters or fewer. How many terms are in the dictionary?

In [ ]:
```
from src.notes import get_cui_list

inflammatory_disorder_list = get_cui_list(snomed_ct_concept_str, "C1290884",

#============
# Sanity Check

if len(inflammatory_disorder_list) != 2991:
    warnings.warn(f"Length of inflammatory_disorder_list differs from expect

if "ekc" != inflammatory_disorder_list[0]:
    warnings.warn(f"First element of inflammatory_disorder_list differs from
```

2.12 (7 pts)

Now let's determine if the notes contain these terms. Implement the function `extract_terms` in `src/notes.py` to search the note text for the terms you collected in the previous step, following the instructions in the docstring. When you are done, run the cell below to sanity check your implementation.

In [ ]:
```
from src.notes import extract_terms

term_df = extract_terms(notes_filtered, inflammatory_disorder_list, 50)

#============
# Sanity Check
if term_df.shape[0] != 13213:
    warnings.warn(f"Number of rows differs from expected: shape[0] = {term_d
```

2.13 (6 pts)

Now that we have extracted the terms from the notes and have a representation of which term is in which note in a `wide` dataframe format, we want to determine which concepts are present in each note. To do this, we will reshape the dataframe to a `long` format and normalize terms back to their corresponding concepts.

Implement the function `normalize_terms` in `src/notes.py` following the instructions in the docstring. When you are done, run the cell below to sanity check your implementation.

In [ ]:
```
from src.notes import normalize_terms

concept_df = normalize_terms(term_df, snomed_ct_concept_str)
```

2.14 (7 pts)

As with the diagnoses, we must transform these concepts data into a patient-feature matrix. Transform `concept_df` into a patient-feature matrix where each row is a patient and each column is the presence or absence of a concept. Here we are not going to do any time binning. Each concept should have only one column. Instead of counts, use a binary indicator to indicate that the concept was present in the patient's notes.

Implement the function `get_note_concept_features` in `src/notes.py` following the instructions in the docstring. When you are done, run the cell below to sanity check your implementation.

In [ ]:
```
from src.notes import get_note_concept_features

note_concept_features = get_note_concept_features(concept_df)
```

2.15 (2 pts)

Now let's engineer some features from vital sign measurements also relevant to predicting septic shock! Load in the `vitals_small_cohort.csv` file by running the cell below.

In [ ]:
```
# Run this cell to load the data from the CSV files into Pandas DataFrames
# Note: You do not need to modify the code in this cell

# Reads in the table from the CSV file
vitals = pd.read_csv(os.path.join(data_dir, "vitals_small_cohort.csv"))

# Preprocess the dates
preprocess_dates(vitals, ["charttime"], ["%Y-%m-%dT%H:%M:%SZ"], inplace=True
```

Let's filter the vitals so we are only looking at Heart Rate measurements that were taken prior to the patient's index time.

Implement the function `filter_vitals` in `src/vitals.py` to filter the vitals dataframe to only include measurements that were taken prior to the patient's index time, following the instructions in the docstring. When you are done, run the cell below to sanity check your implementation.

In [ ]:
```
from src.vitals import filter_vitals

vitals_filtered_hr = filter_vitals(vitals, shock_labels, ["HeartRate"])

# ============
# Sanity Check
```

```
if vitals_filtered_hr.shape[0] != 9328:
    warnings.warn(f"Number of rows differs from expected: shape[0] = {vitals
```

2.16 (4 pts)

Now lets construct some features. One feature of interest might be the latest value of the heart rate before the `index_time`.

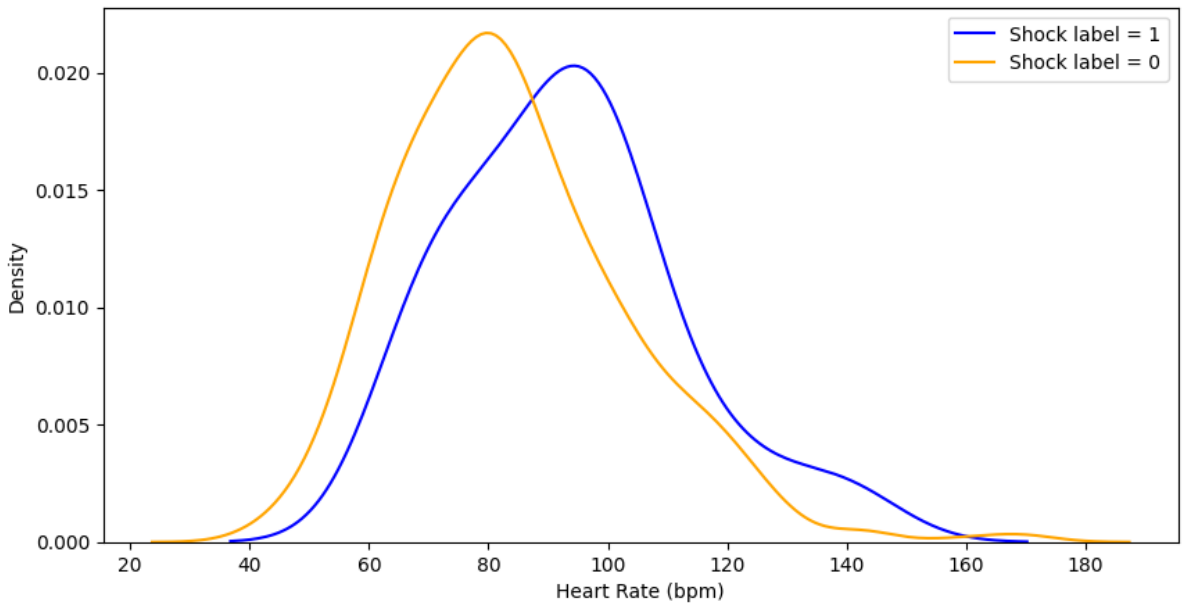
Implement the function `get_latest_hr` in `src/vitals.py` to get the latest heart rate measurement before the `index_time` for each patient. When you are done, run the cell below to sanity check your implementation.

In [ ]:
```
from src.vitals import get_latest_hr

latest_hr_df = get_latest_hr(vitals_filtered_hr)
```

Now, let's create a histogram to look at the distribution of the latest heart rate values.

Implement the function `show_hr_hist` in `src/visualize.py` to plot a histogram of the latest heart rate values, following the instructions in the docstring. When you are done, run the cell below to sanity check your implementation.

In [ ]:
```
from src.visualize import show_hr_plot

# Create the plot
show_hr_plot(latest_hr_df, "hr_plot.png")
```

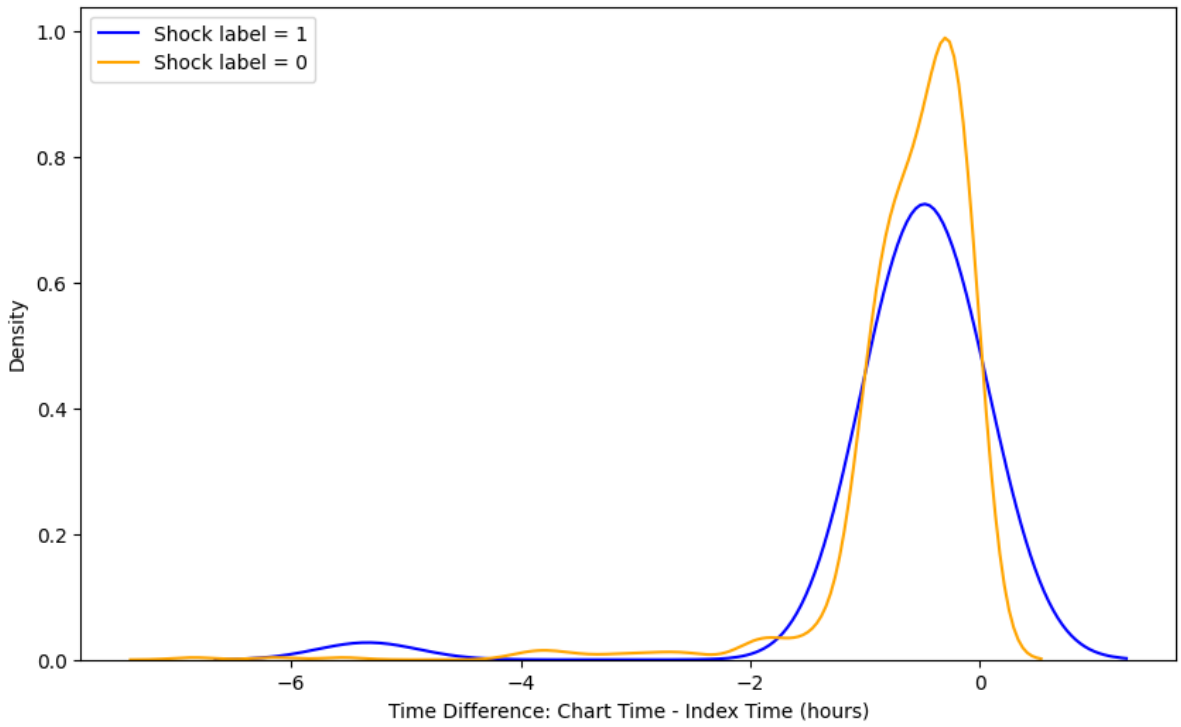


2.17 (4 pts)

There are some additional considerations we should think about prior to utilizing the latest heart rate feature in our model. For example, if the latest recorded heart rate is not very close to the patient's `index_time`, the feature may not be very useful for that patient.

To examine this issue, let's plot the distribution of the time between the latest heart rate measurement and the `index_time`. Implement the function `show_hr_time_hist` in `src/visualize.py` to plot a histogram of the time between the latest heart rate measurement and the `index_time`, following the instructions in the docstring. When you are done, run the cell below to sanity check your implementation.

In [ ]:
```
from src.visualize import show_hr_time_plot

# Create the plot
show_hr_time_plot(latest_hr_df, "hr_plot.png")
```



2.18 (5 pts)

Another concern is that when monitoring patients, especially when thinking about heart rate recordings, relying on a single data point can be misleading. By merely using the last recorded value, we run the risk of using an atypical value. Imagine a scenario where a patient's heart rate is regularly around 80 beats per minute, but due to some temporary distress or a device error, the last recorded value spikes to 120 bpm. If we base our analysis or decisions on this single data point, our conclusions will be skewed.

To address these concerns, instead of using just the last measurement, we can utilize a more robust metric: the time-weighted average heart rate. The idea behind a time-weighted average is to account for all measurements while giving more weight to recent ones. This ensures that:

- All data points contribute to the final value.
- More recent data has a higher influence on the average, as it might be more relevant to the patient's current state.

Use the formula $w = e^{(-|\Delta t|-1)}$ to calculate the weights of each measurement, where Δt is the time difference between the measurement time and the cutoff time in hours.

Calculate the weighted average for each patient with the formula $\bar{x}_w = \sum(x_i w_i) / \sum(w_i)$, where x_i is the value of the measurement and w_i is the weight of that measurement, and i ranges from 1 to the total number of measurements for that patient.

The result should be a dataframe with two columns: `subject_id` and `time_wt_avg`. Implement the function `get_time_weighted_hr` in `src/vitals.py` to calculate the time-weighted average heart rate for each patient, following the instructions in the docstring. When you are done, run the cell below to sanity check your implementation.

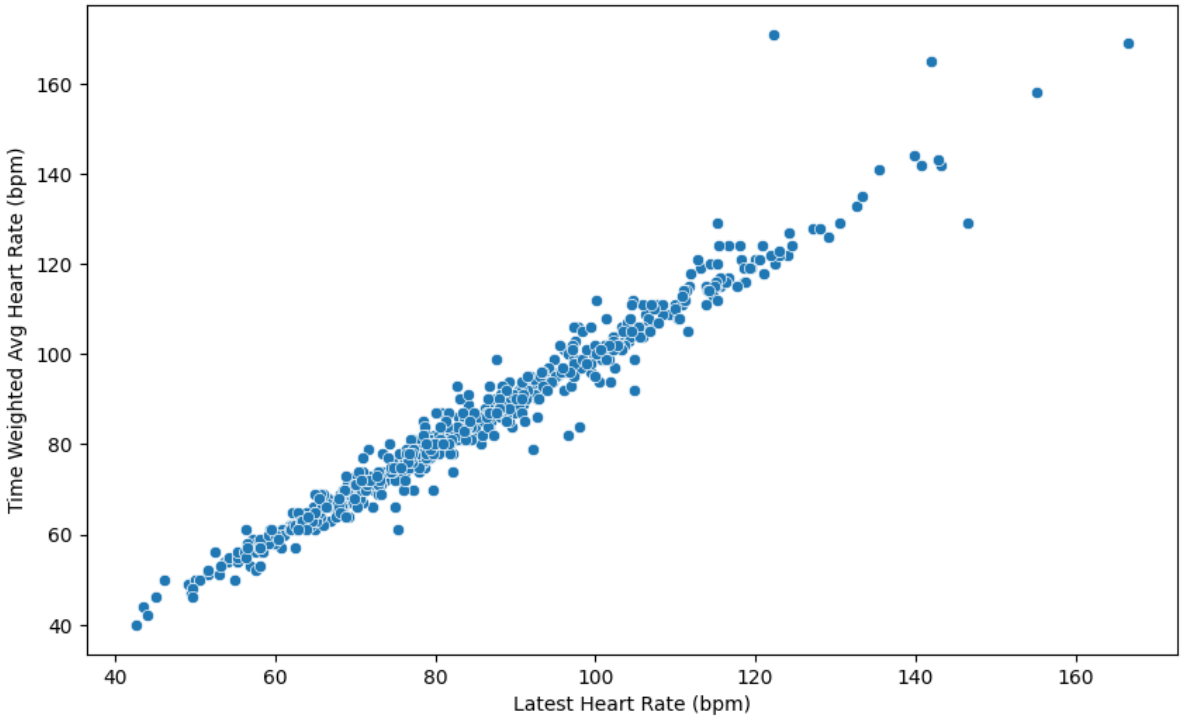
In [ ]:
```
from src.vitals import get_time_weighted_hr

time_weighted_hr_df = get_time_weighted_hr(vitals_filtered_hr)
```

2.19 (4 pts)

Let's do a sanity check to see if what we've done makes sense. We expect that the time-weighted average heart rate and the latest recorded heart rate should be similar.

Make a scatterplot of the latest recorded heart rate (x-axis) and the time-weighted average heart rate (y-axis) of each patient. Implement the function `show_hr_scatter` in `src/visualize.py` to plot the scatterplot. When you are done, run the cell below to sanity check your implementation.

In [ ]:
```
from src.visualize import show_hr_scatter

# Create the plot
show_hr_scatter(latest_hr_df, time_weighted_hr_df, "hr_scatter.png")
```



In [ ]:
```
# Finally run this cell to get all the heart rate features together in the s
heart_rate_features = pd.merge(
        latest_hr_df.drop(columns=["charttime"]),
        time_weighted_hr_df,
        on="subject_id",
        how="inner",
    ).drop(columns=["label", "index_time"])
```

2.20 (4 pts)

We're almost there! Our final patient-feature matrix will simply be the amalgamation of the different feature matrices we've created. Implement `join_and_clean_data` in `src/utils.py` to combine the columns of the feature matrices from diagnoses, notes, and heart rate measurements, following the instructions in the docstring. Note that not all patients have diagnoses or note features, so this function should fill in any NA values with 0 to indicate that there were no diagnoses or notes counted. Similarily, not all subjects have heart rate measurements. Fill NA values for these features with a simple column mean imputation.

In [ ]:
```
from src.utils import join_and_clean_data

joined = join_and_clean_data(diagnosis_features, note_concept_features, hear

#============
# Sanity Check

if joined.shape[0] != 773:
    warnings.warn(f"Number of rows differs from expected: shape[0] = {joinec
```

```
if joined.isna().sum().sum() != 0:
    warnings.warn(f"Dataframe contains NaN values, which is not expected", S
```

Use `list(joined.columns)` to look at all the features and make sure everything makes sense.

How many total features are there?

There are a total of 1169 features (total columns - 1 to take out subject_id)

In [ ]:
```
# TODO: Add code to this cell to answer question above
len(list(joined.columns)) - 1
```

Out[ ]: 1169

3 Open Ended Feature Engineering - Do something cool! (20 pts)

Having made it this far, you have picked up a few generalizable techniques that can now be used to extract features from various modalities of clinical data. To test the skills you've learned thus far, you now have free rein to get creative and derive whatever additional features you would like and use them alongside the disease, text and vitals features as input to a simple classifier. To help you with your task, we provide you with CSV files for ALL of the tables in MIMIC III where each table has been filtered to contain only the records for the patients in our small cohort. These are stored in the folder `additional_data`.

In [ ]:
```
# Run this cell to see what data files are available
# Note: You do not need to modify the code in this cell

# Let's take a look at what files we have available
# Note: You do not need to modify the code in this cell
file_list = os.listdir(os.path.join(data_dir, "additional_data"))
file_list.sort()

print("Available files: ---")
for ind, f in enumerate(file_list):
    print(f, end=" "*10)
    if ind % 3 == 2:
        print()
```

```
Available files: ---
admissions_additional.csv          chartevents_10_additional.csv          c
hartevents_11_additional.csv
chartevents_12_additional.csv          chartevents_13_additional.csv
chartevents_14_additional.csv
chartevents_1_additional.csv          chartevents_2_additional.csv
chartevents_3_additional.csv
chartevents_4_additional.csv          chartevents_5_additional.csv
chartevents_6_additional.csv
chartevents_7_additional.csv          chartevents_8_additional.csv
cptevents_additional.csv
d_cpt_additional.csv          d_icd_diagnoses_additional.csv          d_icd
_procedures_additional.csv
d_items_additional.csv          d_labitems_additional.csv          datetime
events_additional.csv
diagnoses_icd_additional.csv          drgcodes_additional.csv          icus
tays_additional.csv
inputevents_cv_additional.csv          inputevents_mv_additional.csv
labevents_additional.csv
microbiologyevents_additional.csv          noteevents_additional.csv
outputevents_additional.csv
patients_additional.csv          prescriptions_additional.csv          proc
edureevents_mv_additional.csv
procedures_icd_additional.csv          services_additional.csv          tra
nsfers_additional.csv
```

We also provide you with some baseline code below that runs a logistic regression classifier with a Lasso L1 penalty and reports a cross-validation AUC-ROC. Use the code below to see the performance of the model with the features you have already engineered.

In [ ]:
```
# Run this cell (Depending on your computer and your implementation, this ce
# Note: fit_model is a provided function, you do not need to implement it
# Note: Your implementation is not expected to hit any performance targets.
# With only the features we have defined above, note the results are not be
# In future assignments we will take a closer look at models!
from src.model import fit_model
fit_model(joined, shock_labels)
```

```
 20%|██        | 1/5 [00:33<02:12, 33.22s/fold]
Fold 1 ROC AUC Score: 0.5906
 40%|████      | 2/5 [01:04<01:35, 31.80s/fold]
Fold 1 ROC AUC Score: 0.5418
 60%|██████    | 3/5 [01:40<01:08, 34.09s/fold]
Fold 1 ROC AUC Score: 0.4919
 80%|████████  | 4/5 [02:09<00:31, 31.78s/fold]
Fold 1 ROC AUC Score: 0.5625
100%|██████████| 5/5 [02:45<00:00, 33.02s/fold]
Fold 1 ROC AUC Score: 0.5064
Mean ROC AUC Score: 0.5386467124036185
```

Use the code to do the following:

- Outside of the features we engineered previously in the assignment, derive additional features that utilize at **least five of the additional data tables**. You may use tables that we have previously worked with as a part of the assignment, but we encourage you to explore these new data sources. Caveats: definition tables (e.g. d_items) do not count towards the five and using any combination of chartevents tables counts as a single table.
- Combine your derived features into a patient-feature matrix
- Adapt the model-fitting code provided above to your new dataset below

In [ ]:
```
# Let's load in some data
# file_name = "icustays_additional.csv" # <-- TODO: Change this to any file
# additional_data = pd.read_csv(os.path.join(data_dir, "additional_data", fi

# TODO: Repeat the above code for other tables as needed
```

Feature set 1: Patient Age

- Table used: `patients_additional.csv`
- Population with compromised immune system such as new borns and elderly are more susceptible to sepsis and septic shock. This feature aims to capture the age of the patient at the admission used for developing shock_label. Model may learn the association between potential septic_shock onset within 12 hours into admission and the age of patient.
- As age over 89 are shifted to 300 years prior to admission according to MIMIC-III documentation, any age above 89 are manually capped at 89. Since in Assignment 3 cohort building we only included patients with age over 15, we expect that the range of values for this feature to be 15-89.

In [ ]:
```
file_name = "patients_additional.csv"
patient_additional = pd.read_csv(os.path.join(data_dir, "additional_data", f

preprocess_dates(patient_additional, ["dob"], ["%Y-%m-%dT%H:%M:%SZ"],inplace

shock_labels = get_shock_labels(merged_cohort)
```

In [ ]:
```
def get_age_feature(patient_additional, shock_labels):
    merged = shock_labels.merge(patient_additional, on=['subject_id'], how='
    merged['admittime'] = merged['admittime'].dt.date
    merged['dob'] = merged['dob'].dt.date
    merged['subject_age'] = merged['admittime']-merged['dob']
    merged['subject_age'] = merged['subject_age'].apply(lambda x: x.days/365
    merged.loc[(merged['subject_age'] >= 89), 'subject_age'] = 89
    merged = merged[['subject_id', 'subject_age']]

    return merged
```

In [ ]:
```
age_feature = get_age_feature(patient_additional, shock_labels)
```

In [ ]:
```
age_feature[:3]
```

Out[ ]:

	subject_id	subject_age
0	4	47.876712
1	6	65.983562
2	9	41.816438

Feature set 2: Past ICU stays

- Used table: `icustays_additional.csv`
- The selected cohort all underwent septic shock during their admission to the ICU as defined in assignment 3. For patients with repeated ICU admission in this cohort, they may be more susceptible to septic shock in a future admission, and time they spent in ICUs may also relate with their septic shock onset time after admission.
- We calculate the *averaged number of ICU stays per admission* and the *averaged time spent per admission* (in fractional days) in ICUs for these patients.

In [ ]:
```
file_name = "icustays_additional.csv"
icustays_additional = pd.read_csv(os.path.join(data_dir, "additional_data",
```

In [ ]:
```
def get_icustays_features(icustays):
    icustays_additional = icustays.copy()
    icustays_avg = pd.pivot_table(icustays_additional, values=['icustay_id',
                        aggfunc={'icustay_id': "count", 'los': "mean"}).rese
    icustays_avg = pd.pivot_table(icustays_avg, values=['icustay_id', 'los']
                        aggfunc={'icustay_id': "mean", 'los': "mean"}).reset
    icustays_avg.rename(columns={'icustay_id':'icustay_avg_count', 'los':'lo

    return icustays_avg
```

In [ ]:
```
icustays_feature = get_icustays_features(icustays_additional)
```

In [ ]:
```
icustays_feature[:3]
```

Out[ ]:

	subject_id	icustay_avg_count	los_avg
0	3	1.0	6.0646
1	4	1.0	1.6785
2	6	1.0	3.6729

Feature set 3: Related vitals

- Table used: `d_items_additional.csv` + `chartevents csv tables`
- Population with septic shock may experience lowered systolic blood pressure, irregular body temperature and respiratory rates prior to septic shock. Similar to

heart rate, we will compute the latest and time-weighted average values for these vital signs prior to index time for each patientt.

- First, systolic blood pressure and respiratory rate itemids are extracted from `d_items_additional.csv`, and related codes are manually picked to be used later for filtering chartevents data.
- The latest and time-weighted vitals for each patient are calculated in the same manner as the heart rate in section 2 to keep vital features consistent.

In [ ]:
```
file_name = "d_items_additional.csv"
d_items_additional = pd.read_csv(os.path.join(data_dir, "additional_data", f

def filter_itemsid_sysbp(d_items, search_string):
    d_items_additional = d_items.copy()
    pattern = '|'.join(search_string)
    d_items_additional['has_bp']  = d_items_additional['label'].str.contains
    d_items_filtered = d_items_additional[d_items_additional['has_bp']==1]

    return d_items_filtered
```

In [ ]:
```
search_string = ["systolic"]
d_items_filt = filter_itemsid_sysbp(d_items_additional, search_string)
d_items_filt
```

Out[ ]:

	row_id	itemid	label	abbreviation	dbsource	linksto	category	u
295	32	6	ABP [Systolic]	NaN	carevue	chartevents	NaN	
320	57	51	Arterial BP [Systolic]	NaN	carevue	chartevents	NaN	
671	408	442	Manual BP [Systolic]	NaN	carevue	chartevents	NaN	
682	419	455	NBP [Systolic]	NaN	carevue	chartevents	NaN	
705	442	480	Orthostat BP sitting [Systolic]	NaN	carevue	chartevents	NaN	
707	444	482	OrthostatBP standing [Systolic]	NaN	carevue	chartevents	NaN	
709	446	484	Orthostatic BP lying [Systolic]	NaN	carevue	chartevents	NaN	
715	452	492	PAP [Systolic]	NaN	carevue	chartevents	NaN	
1437	618	666	Systolic Unloading	NaN	carevue	chartevents	NaN	
1748	929	3313	BP Cuff [Systolic]	NaN	carevue	chartevents	NaN	
1750	931	3315	BP Left Arm [Systolic]	NaN	carevue	chartevents	NaN	
1752	933	3317	BP Left Leg [Systolic]	NaN	carevue	chartevents	NaN	
1754	935	3319	BP PAL [Systolic]	NaN	carevue	chartevents	NaN	
1756	937	3321	BP Right Arm [Systolic]	NaN	carevue	chartevents	NaN	
1758	939	3323	BP Right Leg [Systolic]	NaN	carevue	chartevents	NaN	
1760	941	3325	BP UAC [Systolic]	NaN	carevue	chartevents	NaN	
4542	4736	7643	RVSYSTOLIC	NaN	carevue	chartevents	NaN	
5062	4325	6701	Arterial BP #2 [Systolic]	NaN	carevue	chartevents	NaN	
9207	15339	228152	Aortic Pressure Signal - Systolic	Aortic Pressure Signal - Systolic	metavision	chartevents	Impella	
9314	13050	224167	Manual Blood	Manual BPs L	metavision	chartevents	Routine Vital Signs	

	row_id	itemid	label	abbreviation	dbsource	linksto	category	u
			Pressure Systolic Left					
9443	14619	227243	Manual Blood Pressure Systolic Right	Manual BPs R	metavision	chartevents	Routine Vital Signs	
9453	14629	226850	RV systolic pressure(PA Line)	RV systolic pressure(PA Line)	metavision	chartevents	PA Line Insertion	
9455	14631	226852	PA systolic pressure(PA Line)	PA systolic pressure(PA Line)	metavision	chartevents	PA Line Insertion	
11503	12716	220050	Arterial Blood Pressure systolic	ABPs	metavision	chartevents	Routine Vital Signs	
11509	12721	220059	Pulmonary Artery Pressure systolic	PAPs	metavision	chartevents	Hemodynamics	
11522	12734	220179	Non Invasive Blood Pressure systolic	NBPs	metavision	chartevents	Routine Vital Signs	
12443	13687	225309	ART BP Systolic	ART BP Systolic	metavision	chartevents	Routine Vital Signs	

In [ ]:
```
## define the systolic bp itemids
itemid_sysbp = [224167, 227243, 220050,220179]
```

In [ ]:
```
search_string = ["respiratory"]
d_items_filt = filter_itemsid_sysbp(d_items_additional, search_string)
d_items_filt
```

Out[ ]:

	row_id	itemid	label	abbreviation	dbsource	linksto	categ
261	573	616	Respiratory Effort	NaN	carevue	chartevents	
262	574	617	Respiratory Pattern	NaN	carevue	chartevents	
263	575	618	Respiratory Rate	NaN	carevue	chartevents	
1395	576	619	Respiratory Rate Set	NaN	carevue	chartevents	
3858	2781	3605	Respiratory Support	NaN	carevue	chartevents	
9375	13933	225475	Respiratory Arrest	Respiratory Arrest	metavision	procedureevents_mv	Signifi Ev
9409	14724	227047	Post-Operative Respiratory (RESPIRAT)	Respiratory Post-Operative	metavision	chartevents	Scor APACH
9581	13012	223990	Respiratory Effort	Respiratory Effort	metavision	chartevents	Pulmo
9701	14709	227032	Non-Operative Respiratory (RESPIRAT)	Respiratory Non-Operative	metavision	chartevents	Scor APACH
10036	13227	223985	Respiratory Pattern	Respiratory Pattern	metavision	chartevents	Pulmo
10053	13244	224688	Respiratory Rate (Set)	Respiratory Rate (Set)	metavision	chartevents	Respira
10054	13245	224689	Respiratory Rate (spontaneous)	Respiratory Rate (spontaneous)	metavision	chartevents	Respira
10055	13246	224690	Respiratory Rate (Total)	Respiratory Rate (Total)	metavision	chartevents	Respira
10663	12303	70057	Rapid Respiratory Viral Screen & Culture	NaN	hospital	microbiologyevents	SPECI
10664	12304	70058	RAPID RESPIRATORY VIRAL ANTIGEN TEST	NaN	hospital	microbiologyevents	SPECI
10704	13284	224745	Respiratory Quotient	RQ	metavision	chartevents	Respira
10924	12516	80177	RESPIRATORY SYNCYTIAL VIRUS (RSV)	NaN	hospital	microbiologyevents	ORGAN
11526	12738	220210	Respiratory	RR	metavision	chartevents	Respira

	row_id	itemid	label	abbreviation	dbsource	linksto	categ
			Rate				

In [ ]:
```
## define the respiratory rate itemids
itemid_resp = [224690, 220210]
```

In [ ]:
```
search_string = ["temperature"]
d_items_filt = filter_itemsid_sysbp(d_items_additional, search_string)
d_items_filt
```

Out[ ]:

	row_id	itemid	label	abbreviation	dbsource	linkst
236	548	591	RLE [Temperature]	NaN	carevue	chartevent
242	554	597	RUE [Temperature]	NaN	carevue	chartevent
1417	598	645	Skin [Temperature]	NaN	carevue	chartevent
1446	627	676	Temperature C	NaN	carevue	chartevent
1447	628	677	Temperature C (calc)	NaN	carevue	chartevent
1448	629	678	Temperature F	NaN	carevue	chartevent
1449	630	679	Temperature F (calc)	NaN	carevue	chartevent
4813	5007	8537	Temp/Iso/Warmer [Temperature, degrees C]	NaN	carevue	chartevent
9306	13042	224027	Skin Temperature	Skin Temp	metavision	chartevent
9416	14731	227054	TemperatureF_ApacheIV	TemperatureF_ApacheIV	metavision	chartevent
10044	13235	224674	Changes in Temperature	Changes in Temperature	metavision	chartevent
11323	13423	224642	Temperature Site	Temp Site	metavision	chartevent
11464	15236	228242	Pt. Temperature (BG) (SOFT)	Pt. Temperature (BG) (SOFT)	metavision	chartevent
12254	14446	226329	Blood Temperature CCO (C)	Blood Temp CCO (C)	metavision	chartevent
12366	12757	223761	Temperature Fahrenheit	Temperature F	metavision	chartevent
12367	12758	223762	Temperature Celsius	Temperature C	metavision	chartevent

In [ ]:
```
## define the temperature itemids
itemid_temp = [223761, 223762]
itemid_temp_units = {
    223761:'F',
    223762:'C'
}
```

In [ ]:
```
# Read all chartevents tables
file_list = os.listdir(os.path.join(data_dir, "additional_data"))
file_list.sort()

file_list_chartevents = [file for file in file_list if 'chartevent' in file]
```

In [ ]:
```
file_dfs = [pd.read_csv(os.path.join(data_dir, "additional_data", file)) for

df_chartevents = pd.concat(file_dfs)
preprocess_dates(df_chartevents, ["charttime"], ["%Y-%m-%dT%H:%M:%SZ"],inpla
```

```
/var/folders/55/cwx6q2h16k1f7mlmfr7md0hc0000gn/T/ipykernel_23717/284993428
9.py:1: DtypeWarning: Columns (8) have mixed types. Specify dtype option on
import or set low_memory=False.
```

In [ ]:
```
import numpy as np

def get_time_weighted_vitals(df_chartevents, itemid_vitals, shock_labels, vi
    df_vitals = df_chartevents[df_chartevents['itemid'].isin(itemid_vitals)]
    result = shock_labels.merge(df_vitals, how='inner', on=['subject_id', 'h
    result = result[result['charttime'] < result['index_time']]

    latest_vital = result.sort_values('charttime')
    latest_vital = latest_vital.groupby('subject_id').tail(1)
    latest_vital = latest_vital[['subject_id', 'valuenum']]
    latest_vital = latest_vital.rename(columns={'valuenum':f'latest_{vital_r

    ## get time weighted feature

    result = result.sort_values('charttime')
    result['dt'] = (result['charttime'] - result['index_time'])/ pd.Timedelt
    result['weight'] = np.exp(-1*abs(result['dt']) - 1)
    result['weighted_vital'] = result['weight']*result['valuenum']
    result = result[['subject_id', 'dt', 'weight', 'weighted_vital', 'valuer
    time_weighted_vital = result.groupby('subject_id', as_index=False).sum()
    time_weighted_vital[f'time_wt_avg_{vital_name}'] = time_weighted_vital['
    time_weighted_vital = time_weighted_vital[['subject_id', f'time_wt_avg_{
    time_weighted_vital = time_weighted_vital.dropna()

    vital_features = latest_vital.merge(time_weighted_vital, how='outer', or

    return vital_features
```

In [ ]:
```
## temperature readings need to be converted to the same unit
def get_time_weighted_temp(df_chartevents, itemid_vitals, itemid_units, shoc
    df_vitals = df_chartevents[df_chartevents['itemid'].isin(itemid_vitals)]
    df_vitals['unit'] = df_vitals['itemid'].apply(lambda x: itemid_units[x])
    df_vitals.loc[df_vitals['unit']=='F', 'valuenum'] = (df_vitals[df_vitals
    result = shock_labels.merge(df_vitals, how='inner', on=['subject_id', 'h
    result = result[result['charttime'] < result['index_time']]

    latest_vital = result.sort_values('charttime')
    latest_vital = latest_vital.groupby('subject_id').tail(1)
    latest_vital = latest_vital[['subject_id', 'valuenum']]
    latest_vital = latest_vital.rename(columns={'valuenum':f'latest_{vital_r
```

```
    ## get time weighted feature
    result = result.sort_values('charttime')
    result['dt'] = (result['charttime'] - result['index_time'])/ pd.Timedelt
    result['weight'] = np.exp(-1*abs(result['dt']) - 1)
    result['weighted_vital'] = result['weight']*result['valuenum']
    result = result[['subject_id', 'dt', 'weight', 'weighted_vital', 'valuer
    time_weighted_vital = result.groupby('subject_id', as_index=False).sum()
    time_weighted_vital[f'time_wt_avg_{vital_name}'] = time_weighted_vital['
    time_weighted_vital = time_weighted_vital[['subject_id', f'time_wt_avg_{
    time_weighted_vital = time_weighted_vital.dropna()

    vital_features = latest_vital.merge(time_weighted_vital, how='outer', or

    return vital_features
```

In [ ]:
```
sysbp_features = get_time_weighted_vitals(df_chartevents, itemid_sysbp, shoc
resp_features = get_time_weighted_vitals(df_chartevents, itemid_resp, shock_
temp_features = get_time_weighted_temp(df_chartevents, itemid_temp, itemid_t
```

```
/var/folders/55/cwx6q2h16k1f7mlmfr7md0hc0000gn/T/ipykernel_23717/363992403
4.py:4: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-doc
s/stable/user_guide/indexing.html#returning-a-view-versus-a-copy
```

In [ ]:
```
sysbp_features[:3]
```

Out[ ]:

	subject_id	latest_sysbp	time_wt_avg_sysbp
0	291	102.0	111.272459
1	518	152.0	147.332891
2	904	137.0	137.058476

In [ ]:
```
resp_features[:3]
```

Out[ ]:

	subject_id	latest_resp	time_wt_avg_resp
0	291	14.0	13.878229
1	518	16.0	15.424841
2	904	25.0	26.498720

In [ ]:
```
temp_features[:3]
```

Out[ ]:

	subject_id	latest_temp	time_wt_avg_temp
0	291	35.944444	35.920270
1	518	36.166667	36.260856
2	904	36.833333	36.824312

Feature set 4: Related microbio lab events

- Table used: `microbiologyevents_additional.csv`
- Bacterial infections cause most cases of sepsis. Blood cultures are used to aid in the diagnosis of patients with suspected sepsis secondary to either a fungemia or bacteremia. Positive cultures may relate to septic shock in the future.
- Blood and urine samples are often used for microbiological infection tests. The following specimens are used for generating features here.
- The feature is the count of unique positive organisms that grew in the culture. This indicates the number of unique type of infections. Count = 0 indicates negative / no infections.

In [ ]:
```
file_name = "microbiologyevents_additional.csv"
microbio_additional = pd.read_csv(os.path.join(data_dir, "additional_data",
```

In [ ]:
```
def get_blood_culture_pos_counts(microbio, spec_type_desc):
    microbio_additional = microbio.copy()
    ## filter for blood culture results and merged on shock labels
    merged_microb = shock_labels.merge(microbio_additional[microbio_additior
    ## filter for blood culture results obtained prior to predict time of se
    merged_microb = merged_microb[merged_microb['chartdate']<merged_microb['
    merged_microb_count_pos= merged_microb['org_itemid'].fillna('NEG CULTURE
    merged_microb_count_pos = merged_microb.groupby('subject_id').nunique().
    merged_microb_count_pos = merged_microb_count_pos[['subject_id', 'org_it
    merged_microb_count_pos = merged_microb_count_pos.rename(columns={'org_i

    return merged_microb_count_pos
```

In [ ]:
```
micro_blood_feature = get_blood_culture_pos_counts(microbio_additional, spec
micro_urine_feature = get_blood_culture_pos_counts(microbio_additional, spec
```

In [ ]:
```
micro_blood_feature[:3]
```

Out[ ]:

	subject_id	pos_BLOOD CULTURE_count
0	4	1
1	9	1
2	21	1

In [ ]:
```
micro_urine_feature[:3]
```

Out[ ]:

	subject_id	pos_URINE_count
0	4	0
1	19	0
2	21	0

Feature set 5: Prescriptions prior to index time

- Table used: `prescriptions_additional.csv`
- To treat sepsis, medications such as vasopressin/norepinephrine cause your blood vessels to narrow and increase the blood flow to your organs and are often prescribed. Patient may receive insulin if the septic shock has increased your blood sugar (glucose) levels.
- Average count of the times the above medications are prescribed prior to index time are calculated as new features.

In [ ]:
```
file_name = "prescriptions_additional.csv"
prescriptions_additional = pd.read_csv(os.path.join(data_dir, "additional_da
```

```
/var/folders/55/cwx6q2h16k1f7mlmfr7md0hc0000gn/T/ipykernel_23717/751712753.
py:2: DtypeWarning: Columns (11) have mixed types. Specify dtype option on
import or set low_memory=False.
```

In [ ]:
```
def get_prescriptions_features(prescriptions, drug_target):
    prescriptions_additional = prescriptions.copy()
    ## filter for  and merged on shock labels
    prescriptions_merged = shock_labels.merge(prescriptions_additional[presc
    ## filter for blood culture results obtained prior to predict time of se
    prescriptions_merged = prescriptions_merged[prescriptions_merged['startc
    prescriptions_merged = prescriptions_merged.groupby('subject_id').nuniqu
    prescriptions_merged = prescriptions_merged[['subject_id', 'dose_val_rx'
    prescriptions_merged = prescriptions_merged.rename(columns={'dose_val_rx

    return prescriptions_merged
```

In [ ]:
```
prescrip_feature = get_prescriptions_features(prescriptions_additional, 'Nor
prescrip_feature = prescrip_feature.merge(get_prescriptions_features(prescri
prescrip_feature = prescrip_feature.merge(get_prescriptions_features(prescri
```

In [ ]:
```
prescrip_feature[:3]
```

Out[ ]:

	subject_id	Norepinephrine_prescribed_count	Vasopressin_prescribed_count	Insulin_pres
0	21	1.0	NaN	
1	25	1.0	NaN	
2	62	1.0	1.0	

Combine features

- Vitals related features are imputed with mean of the column. Age feature is also imputed with mean of the column as age of 0 is a misleading value.
- All other features are filled with 0 at NaN locations.

In [ ]:
```
## New features: age_feature, icustays_feature, sysbp_features, temp_feature
all_subject_ids = pd.DataFrame(shock_labels['subject_id'])
new_features_fillzero = all_subject_ids.merge(icustays_feature, how='left',
new_features_fillzero = new_features_fillzero.merge(micro_blood_feature, how
new_features_fillzero = new_features_fillzero.merge(micro_urine_feature, how
new_features_fillzero = new_features_fillzero.merge(prescrip_feature, how='l
new_features_fillzero = new_features_fillzero.fillna(0)
```

In [ ]:
```
new_features_fillzero[:5]
```

Out[ ]:

	subject_id	icustay_avg_count	los_avg	pos_BLOOD CULTURE_count	pos_URINE_count	Norepinephri
0	4	1.0	1.6785	1.0	0.0	
1	6	1.0	3.6729	0.0	0.0	
2	9	1.0	5.3231	1.0	0.0	
3	11	1.0	1.5844	0.0	0.0	
4	12	1.0	7.6348	0.0	0.0	

In [ ]:
```
all_subject_ids = pd.DataFrame(shock_labels['subject_id'])
new_features_fillmean = all_subject_ids.merge(age_feature, how='left', on='s
new_features_fillmean = new_features_fillmean.merge(temp_features, how='left
new_features_fillmean = new_features_fillmean.merge(sysbp_features, how='lef
new_features_fillmean = new_features_fillmean.merge(resp_features, how='left

feature_cols = [x for x in new_features_fillmean.columns if x != 'subject_ic
new_features_fillmean[feature_cols] = new_features_fillmean[feature_cols].fi
```

In [ ]:
```
new_features_fillmean[:5]
```

Out[ ]:

	subject_id	subject_age	latest_temp	time_wt_avg_temp	latest_sysbp	time_wt_avg_sysl
0	4	47.876712	36.730994	36.732153	118.641026	117.9937
1	6	65.983562	36.730994	36.732153	118.641026	117.9937
2	9	41.816438	36.730994	36.732153	118.641026	117.9937
3	11	50.180822	36.730994	36.732153	118.641026	117.9937
4	12	72.419178	36.730994	36.732153	118.641026	117.9937

In [ ]:
```
new_features = new_features_fillmean.merge(new_features_fillzero, on='subjec
```

In [ ]:
```
new_features.shape[1]-1
```

Out[ ]: 14

14 new features are added.

In [ ]:
```
all_features = new_features.merge(joined, on='subject_id', how='inner')
```

In [ ]:
```
all_features
```

Out[ ]:

	subject_id	subject_age	latest_temp	time_wt_avg_temp	latest_sysbp	time_wt_avg_s
0	4	47.876712	36.730994	36.732153	118.641026	117.99
1	9	41.816438	36.730994	36.732153	118.641026	117.99
2	11	50.180822	36.730994	36.732153	118.641026	117.99
3	13	39.890411	36.730994	36.732153	118.641026	117.99
4	17	47.849315	36.730994	36.732153	118.641026	117.99
...	...	...	...	...	...	
768	1382	41.945205	36.730994	36.732153	118.641026	117.99
769	1385	70.298630	36.730994	36.732153	118.641026	117.99
770	1386	55.978082	36.730994	36.732153	118.641026	117.99
771	1387	48.726027	36.730994	36.732153	118.641026	117.99
772	1390	84.402740	36.730994	36.732153	118.641026	117.99

773 rows × 1184 columns

In [ ]:
```
fit_model(all_features, shock_labels) # TODO: Uncomment this line to run you
```

```
  0%|          | 0/5 [00:00<?, ?fold/s] 20%|██        | 1/5 [00:13<00:52, 1
3.08s/fold]
Fold 1 ROC AUC Score: 0.7914
 40%|████      | 2/5 [00:28<00:42, 14.20s/fold]
Fold 1 ROC AUC Score: 0.6168
 60%|██████    | 3/5 [00:48<00:33, 16.87s/fold]
Fold 1 ROC AUC Score: 0.6509
 80%|████████  | 4/5 [01:07<00:17, 17.74s/fold]
Fold 1 ROC AUC Score: 0.6985
100%|██████████| 5/5 [01:19<00:00, 15.98s/fold]
Fold 1 ROC AUC Score: 0.8190
Mean ROC AUC Score: 0.7153299738935098
```

Write 1-2 paragraphs discussing what and how many features you derived. Additionally, discuss the effects of those features on the performance of the classifier.

In part 1 and 2 of the assignment, the features developed are the following categories:

- Count of Past diagnosis codes in 6 months prior to index time or > 6 months prior to index time - using the highly specific/informative ICD-9 codes with IC between 4-9
- Notes: using search terms related to inflammatory disorders within current admission's diagnosis notes
- Vitals: latest heart rate for a single patient and time weighted heart rate among all admissions for the patient.

For new features, I used information related to sepsis diagnosis/treatment to design features that are most specific/content rich for the model for septic shock onset prediction. Specifically, the following information are used to engineer the features (more details on feature calculation explained in respective sections above):

- Age: Population with compromised immune system such as new borns and elderly are more susceptible to sepsis and septic shock. This feature aims to capture the age of the patient at the admission used for determining shock_label.
- ICU stays: number of times this patient went into ICU and the mean length of time stayed in ICU. These may indicate compromised immune system which could relate to higher rate of septic shock in future admissions. Averaged number of ICU stays per admission and the averaged time spent per admission (in fractional days) in ICUs are calculated for each patient.
- Related vitals: Vitals signs that relate to septic shock (other than heart rate) include: fever/hypothermia, hyperventilation, shortness of breath or low blood pressure. Hence, latest and time-weighted features for body temperature, respiratory rate and systolic blood pressure prior to index time are included as new features.
- Related microbial test results: sepsis happens when the body's immune system has an extreme response to an infection. Information regarding microbial infection can be informative for sepsis shock prediction. Hence, the count of unique blood and urine culture infections are calculated as new features.
- Prescription: medications such as vasopressin/norepinephrine cause blood vessels to narrow and increase the blood flow to organs and are often prescribed to sepsis patients. Patient may receive insulin if the septic shock has increased your blood sugar (glucose) levels. The average count of the times the above medications prescribed prior to index time are calculated as new features.

Before adding these features, the mean ROC AUC Score was 0.538. After adding the new features to the training process, the mean ROC AUC score raised to 0.715. The new features increased performance of the classifier, and suggests that they are information-rich features that heavily impacted the model for septic shock predictions.

Feedback (`0 points`)

Please fill out the following feedback form so we can improve the course for future students!

Submission Instructions ✅

There are two files you must submit for this assignment:

1. A `PDF` of this notebook.

- **Please clear any large cell outputs from executed code cells before creating the PDF.**
  - Including short printouts is fine, but please try to clear any large outputs such as dataframe printouts. This makes it easier for us to grade your assignments!
- To export the notebook to PDF, you may need to first create an HTML version, and then convert it to PDF.

2. A `zip` file containing your code generated by the provided `create_submission_zip.py` script:

- Open the `create_submission_zip.py` file and enter your SUNet ID where indicated.
- Run the script via `python create_submission_zip.py` to generate a file titled `<your_SUNetID>_submission_A4.zip` in the root project directory.