# Sepsis data - Clustering

Benjamin Frost 2022

In [ ]:
```
import pandas as pd
import numpy as np
import torch.multiprocessing as mp
from sklearn.preprocessing import KBinsDiscretizer, OneHotEncoder, MinMaxScaler
from matplotlib import pyplot as plt
from sklearn.cluster import AgglomerativeClustering
from sklearn.metrics import silhouette_score
from sklearn.metrics import f1_score
from sklearn.metrics import recall_score
from sklearn.metrics import precision_score
from sklearn.cluster import KMeans
from sklearn.preprocessing import PolynomialFeatures
from tqdm import tqdm
from concurrent.futures import ThreadPoolExecutor, as_completed
from scipy.interpolate import interp1d
import random
import Categorization
import torch
import copy
from torch.nn.functional import one_hot
import imblearn
from collections import Counter
from tslearn.clustering import TimeSeriesKMeans, silhouette_score
from tslearn.utils import to_time_series_dataset
from tslearn.preprocessing import TimeSeriesScalerMeanVariance
from tsfresh import extract_features, select_features
from tsfresh.utilities.dataframe_functions import impute
from dask.dataframe import from_pandas
from tsfresh.utilities.distribution import MultiprocessingDistributor
import hashlib
from sklearn.metrics import precision_recall_fscore_support
from importlib import reload
from temporalHelper import TemporalHelper as TH
from concurrent.futures import ProcessPoolExecutor
from PIL import Image
import os
```

## Loading in the data

In [ ]:
```
sepsisDF = pd.read_csv('../LEN_Test/data/sepsis_data.csv')

sepsisDF
```

Out[ ]:

| | Patient_id | HR | O2Sat | Temp | SBP | MAP | DBP | Resp | EtCO2 | BaseExcess | ... | WBC | Fibrinogen | Platelets | Age | Gender | Uni |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| **0** | p116812 | NaN | NaN | NaN | NaN | NaN | NaN | NaN | NaN | NaN | ... | NaN | NaN | NaN | 59.00 | 1 | |
| **1** | p116812 | 102.0 | 100.0 | NaN | NaN | NaN | NaN | 22.0 | NaN | NaN | ... | NaN | NaN | NaN | 59.00 | 1 | |
| **2** | p116812 | 102.0 | 100.0 | NaN | 99.0 | 84.0 | 76.0 | 18.5 | NaN | NaN | ... | NaN | NaN | NaN | 59.00 | 1 | |
| **3** | p116812 | 124.0 | 100.0 | NaN | 97.0 | 70.0 | 55.0 | 16.0 | NaN | NaN | ... | NaN | NaN | NaN | 59.00 | 1 | |
| **4** | p116812 | 98.0 | 100.0 | NaN | 95.0 | 73.0 | 62.0 | 18.0 | NaN | NaN | ... | 6.8 | NaN | 276.0 | 59.00 | 1 | |
| **...** | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... |
| **1552205** | p005863 | 86.0 | 97.0 | NaN | 121.0 | 80.0 | 58.0 | 21.0 | NaN | NaN | ... | NaN | NaN | NaN | 66.47 | 1 | |
| **1552206** | p005863 | 86.0 | 96.0 | NaN | 112.0 | 73.0 | 53.0 | 18.0 | NaN | NaN | ... | NaN | NaN | NaN | 66.47 | 1 | |
| **1552207** | p005863 | 82.0 | 95.0 | NaN | 102.0 | 72.0 | 55.0 | 18.0 | NaN | NaN | ... | NaN | NaN | NaN | 66.47 | 1 | |
| **1552208** | p005863 | 80.0 | 96.0 | 36.11 | 86.0 | 60.0 | 46.0 | 20.0 | NaN | NaN | ... | NaN | NaN | NaN | 66.47 | 1 | |
| **1552209** | p005863 | 84.0 | 94.0 | NaN | 91.0 | 62.0 | 47.0 | 20.0 | NaN | 0.0 | ... | NaN | NaN | NaN | 66.47 | 1 | |

1552210 rows × 42 columns

## Investigating data

In [ ]:
```
# Too many columns to display all in one cell.

step = 10
```

```
for idx in range(0, len(sepsisDF.columns), step):
    tempCols = sepsisDF[sepsisDF.columns[idx:idx+step]]
    display(tempCols.describe())
```

| | HR | O2Sat | Temp | SBP | MAP | DBP | Resp | EtCO2 | BaseExcess |
|---|---|---|---|---|---|---|---|---|---|
| **count** | 1.398811e+06 | 1.349474e+06 | 525226.000000 | 1.325945e+06 | 1.358940e+06 | 1.065656e+06 | 1.313875e+06 | 57636.000000 | 84145.000000 |
| **mean** | 8.458144e+01 | 9.719395e+01 | 36.977228 | 1.237505e+02 | 8.240010e+01 | 6.383056e+01 | 1.872650e+01 | 32.957657 | -0.689919 |
| **std** | 1.732524e+01 | 2.936924e+00 | 0.770014 | 2.323156e+01 | 1.634175e+01 | 1.395601e+01 | 5.098194e+00 | 7.951662 | 4.294297 |
| **min** | 2.000000e+01 | 2.000000e+01 | 20.900000 | 2.000000e+01 | 2.000000e+01 | 2.000000e+01 | 1.000000e+00 | 10.000000 | -32.000000 |
| **25%** | 7.200000e+01 | 9.600000e+01 | 36.500000 | 1.070000e+02 | 7.100000e+01 | 5.400000e+01 | 1.500000e+01 | 28.000000 | -3.000000 |
| **50%** | 8.350000e+01 | 9.800000e+01 | 37.000000 | 1.210000e+02 | 8.000000e+01 | 6.200000e+01 | 1.800000e+01 | 33.000000 | 0.000000 |
| **75%** | 9.550000e+01 | 9.950000e+01 | 37.500000 | 1.380000e+02 | 9.200000e+01 | 7.200000e+01 | 2.150000e+01 | 38.000000 | 1.000000 |
| **max** | 2.800000e+02 | 1.000000e+02 | 50.000000 | 3.000000e+02 | 3.000000e+02 | 3.000000e+02 | 1.000000e+02 | 100.000000 | 100.000000 |

| | HCO3 | FiO2 | pH | PaCO2 | SaO2 | AST | BUN | Alkalinephos | Calcium | 7 |
|---|---|---|---|---|---|---|---|---|---|---|
| **count** | 65028.000000 | 129365.000000 | 107573.000000 | 86301.000000 | 53561.000000 | 25183.000000 | 106568.000000 | 24941.000000 | 91331.000000 | |
| **mean** | 24.075481 | 0.554839 | 7.378934 | 41.021869 | 92.654188 | 260.223385 | 23.915452 | 102.483661 | 7.557531 | |
| **std** | 4.376504 | 11.123207 | 0.074568 | 9.267242 | 10.892986 | 855.746795 | 19.994317 | 120.122746 | 2.433152 | |
| **min** | 0.000000 | -50.000000 | 6.620000 | 10.000000 | 23.000000 | 3.000000 | 1.000000 | 7.000000 | 1.000000 | |
| **25%** | 22.000000 | 0.400000 | 7.340000 | 35.000000 | 94.000000 | 22.000000 | 12.000000 | 54.000000 | 7.700000 | |
| **50%** | 24.000000 | 0.500000 | 7.380000 | 40.000000 | 97.000000 | 41.000000 | 17.000000 | 74.000000 | 8.300000 | |
| **75%** | 26.800000 | 0.600000 | 7.430000 | 45.000000 | 98.000000 | 111.000000 | 28.000000 | 108.000000 | 8.700000 | |
| **max** | 55.000000 | 4000.000000 | 7.930000 | 100.000000 | 100.000000 | 9961.000000 | 268.000000 | 3833.000000 | 27.900000 | |

| | Creatinine | Bilirubin_direct | Glucose | Lactate | Magnesium | Phosphate | Potassium | Bilirubin_total | TroponinI |
|---|---|---|---|---|---|---|---|---|---|
| **count** | 94616.000000 | 2990.000000 | 265516.000000 | 41446.000000 | 97951.000000 | 62301.000000 | 144525.000000 | 23141.000000 | 14781.000000 |
| **mean** | 1.510699 | 1.836177 | 136.932283 | 2.646666 | 2.051450 | 3.544238 | 4.135528 | 2.114059 | 8.290099 |
| **std** | 1.805603 | 3.694082 | 51.310728 | 2.526214 | 0.397898 | 1.423286 | 0.642150 | 4.311468 | 24.806235 |
| **min** | 0.100000 | 0.010000 | 10.000000 | 0.200000 | 0.200000 | 0.200000 | 1.000000 | 0.100000 | 0.010000 |
| **25%** | 0.700000 | 0.200000 | 106.000000 | 1.260000 | 1.800000 | 2.600000 | 3.700000 | 0.500000 | 0.040000 |
| **50%** | 0.940000 | 0.445000 | 127.000000 | 1.800000 | 2.000000 | 3.300000 | 4.100000 | 0.900000 | 0.300000 |
| **75%** | 1.430000 | 1.700000 | 153.000000 | 3.000000 | 2.200000 | 4.100000 | 4.400000 | 1.700000 | 3.980000 |
| **max** | 46.600000 | 37.500000 | 988.000000 | 31.000000 | 9.800000 | 18.800000 | 27.500000 | 49.600000 | 440.000000 |

| | Hgb | PTT | WBC | Fibrinogen | Platelets | Age | Gender | Unit1 | Unit2 |
|---|---|---|---|---|---|---|---|---|---|
| **count** | 114591.000000 | 45699.000000 | 99447.000000 | 10242.000000 | 92209.000000 | 1.552210e+06 | 1.552210e+06 | 940250.000000 | 940250.000000 |
| **mean** | 10.430833 | 41.231193 | 11.446405 | 287.385706 | 196.013911 | 6.200947e+01 | 5.592690e-01 | 0.496571 | 0.503429 |
| **std** | 1.968661 | 26.217669 | 7.731013 | 153.002908 | 103.635366 | 1.638622e+01 | 4.964749e-01 | 0.499989 | 0.499989 |
| **min** | 2.200000 | 12.500000 | 0.100000 | 34.000000 | 1.000000 | 1.400000e+01 | 0.000000e+00 | 0.000000 | 0.000000 |
| **25%** | 9.100000 | 27.800000 | 7.600000 | 184.000000 | 126.000000 | 5.168000e+01 | 0.000000e+00 | 0.000000 | 0.000000 |
| **50%** | 10.300000 | 32.400000 | 10.300000 | 250.000000 | 181.000000 | 6.400000e+01 | 1.000000e+00 | 0.000000 | 1.000000 |
| **75%** | 11.700000 | 42.800000 | 13.800000 | 349.000000 | 244.000000 | 7.400000e+01 | 1.000000e+00 | 1.000000 | 1.000000 |
| **max** | 32.000000 | 250.000000 | 440.000000 | 1760.000000 | 2322.000000 | 1.000000e+02 | 1.000000e+00 | 1.000000 | 1.000000 |

| | ICULOS | SepsisLabel |
|---|---|---|
| **count** | 1.552210e+06 | 1.552210e+06 |
| **mean** | 2.699499e+01 | 1.798468e-02 |
| **std** | 2.900542e+01 | 1.328956e-01 |
| **min** | 1.000000e+00 | 0.000000e+00 |

| | ICULOS | SepsisLabel |
|---|---|---|
| **25%** | 1.100000e+01 | 0.000000e+00 |
| **50%** | 2.100000e+01 | 0.000000e+00 |
| **75%** | 3.400000e+01 | 0.000000e+00 |
| **max** | 3.360000e+02 | 1.000000e+00 |

In [ ]:
```
print(f"There are {sepsisDF['Patient_id'].nunique()} unique patients in the dataset")
```

```
There are 40336 unique patients in the dataset
```

In [ ]:
```
print(sepsisDF['SepsisLabel'].value_counts())
```

```
0    1524294
1      27916
Name: SepsisLabel, dtype: int64
```

## Splitting data by patient

In [ ]:
```
th = TH()

staticColumns = ["Age", "Gender", "Unit1", "Unit2", "HospAdmTime", "ICULOS"]

patients = th.get_patients(sepsisDF, by="Patient_id", label="SepsisLabel", static = staticColumns)
```

```
100%|██████████| 40336/40336 [01:25<00:00, 473.71it/s]
```

## Sanity check

In [ ]:
```
patients[0].data.head()
```

Out[ ]:

| | HR | O2Sat | Temp | SBP | MAP | DBP | Resp | EtCO2 | BaseExcess | HCO3 | ... | Phosphate | Potassium | Bilirubin_total | TroponinI | Hct | Hg |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| **0** | NaN | NaN | NaN | NaN | NaN | NaN | NaN | NaN | NaN | NaN | ... | NaN | NaN | NaN | NaN | NaN | Na |
| **1** | 97.0 | 95.0 | NaN | 98.0 | 75.33 | NaN | 19.0 | NaN | NaN | NaN | ... | NaN | NaN | NaN | NaN | NaN | Na |
| **2** | 89.0 | 99.0 | NaN | 122.0 | 86.00 | NaN | 22.0 | NaN | NaN | NaN | ... | NaN | NaN | NaN | NaN | NaN | Na |
| **3** | 90.0 | 95.0 | NaN | NaN | NaN | NaN | 30.0 | NaN | 24.0 | NaN | ... | NaN | NaN | NaN | NaN | NaN | Na |
| **4** | 103.0 | 88.5 | NaN | 122.0 | 91.33 | NaN | 24.5 | NaN | NaN | NaN | ... | NaN | NaN | NaN | NaN | NaN | Na |

5 rows × 34 columns

In [ ]:
```
patients[0].static.head()
```

Out[ ]:

| | Age | Gender | Unit1 | Unit2 | HospAdmTime | ICULOS |
|---|---|---|---|---|---|---|
| **786527** | 83.14 | 0 | NaN | NaN | -0.03 | 1 |
| **786528** | 83.14 | 0 | NaN | NaN | -0.03 | 2 |
| **786529** | 83.14 | 0 | NaN | NaN | -0.03 | 3 |
| **786530** | 83.14 | 0 | NaN | NaN | -0.03 | 4 |
| **786531** | 83.14 | 0 | NaN | NaN | -0.03 | 5 |

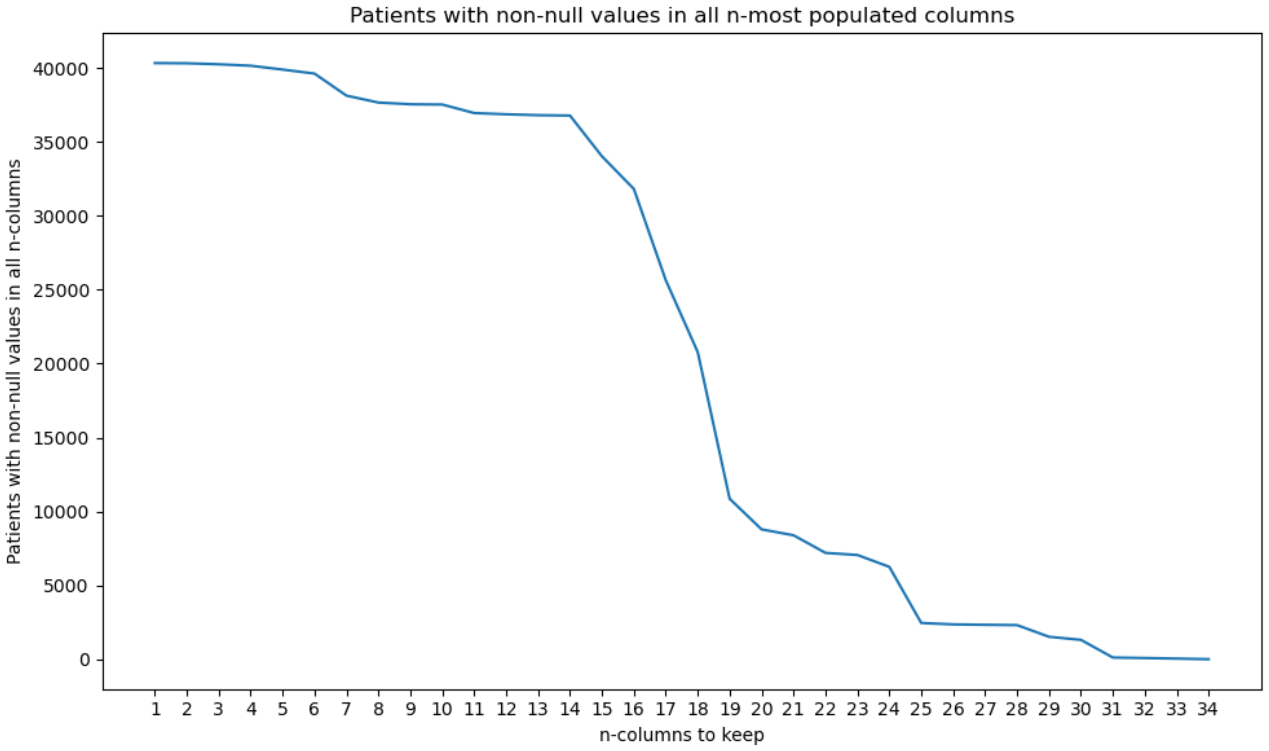
In [ ]:
```
totalNullColumns = 0

for patient in tqdm(patients):
    totalNullColumns += patient.data.isnull().all().sum()

totalColumns = len(patient.data.columns) * len(patients)

print(totalColumns, totalNullColumns)

print(f"{np.round(totalNullColumns / totalColumns * 100, 2)}% of columns are null")
```

```
100%|██████████| 40336/40336 [00:18<00:00, 2224.69it/s]
1371424 477941
34.85% of columns are null
```

## Counting missingness

In [ ]:
```
patientsKept, columnsExplored = th.count_null(patients)
```

```
[1/4] Counting null values...
100%|██████████| 40336/40336 [00:22<00:00, 1810.32it/s]
[2/3] Copying dataset...
[3/4] Dropping null columns...
100%|██████████| 34/34 [01:37<00:00,  2.88s/it]
[4/4] Graphing...
```




## Sharp drop off after 15 columns so will keep around 34000 patients with at least some data in the top 15 columns

In [ ]:
```
clusteringPatients = th.get_top_columns(patients, 15)

print(len(clusteringPatients))

clusteringPatients[3].topColumns.head()
```

```
100%|██████████| 40336/40336 [00:15<00:00, 2532.43it/s]
34028
```

Out[ ]:

| | HR | O2Sat | Resp | MAP | SBP | Temp | Glucose | Potassium | BUN | Creatinine | Hct | Hgb | Platelets | WBC | Magnesium |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| **0** | NaN | NaN | NaN | NaN | NaN | NaN | NaN | NaN | NaN | NaN | NaN | NaN | NaN | NaN | NaN |
| **1** | 103.5 | 97.0 | 18.0 | 70.5 | 107.5 | NaN | NaN | NaN | 14.0 | 0.8 | 27.6 | NaN | 220.0 | NaN | 1.7 |
| **2** | 108.0 | 98.5 | 19.5 | 82.0 | 124.5 | 36.78 | 253.0 | 5.0 | NaN | NaN | NaN | NaN | NaN | NaN | NaN |
| **3** | 107.5 | 96.5 | 17.0 | 77.5 | 117.5 | NaN | NaN | NaN | NaN | NaN | NaN | NaN | NaN | NaN | NaN |
| **4** | 113.0 | 100.0 | 26.0 | 80.0 | 125.0 | NaN | NaN | NaN | NaN | NaN | NaN | NaN | NaN | NaN | NaN |

## Sanity Check

In [ ]:
```
totalNullColumns = 0

for patient in tqdm(clusteringPatients):
    totalNullColumns += patient.topColumns.isnull().all().sum()

totalColumns = len(patient.topColumns.columns) * len(clusteringPatients)

print(totalColumns, totalNullColumns)

print(f"{np.round(totalNullColumns / totalColumns * 100, 2)}% of columns are null")
```

```
100%|██████████| 34028/34028 [00:16<00:00, 2115.63it/s]
510420 0
```

```
0.0% of columns are null
```

## Interpolating missing data

In [ ]:

```
noInterpolation = 0
failureExample = (0,0)

maxPatientLen = 0

tempCP = copy.deepcopy(clusteringPatients)


# Finding the longest patient's time series. Other patients will be extended to match this length.

for idx, patient in (enumerate(tempCP)):

    if patient.data.shape[0] > maxPatientLen:
        maxPatientLen = patient.data.shape[0]

print("Max time series length: " + str(maxPatientLen))


for patient in tqdm(tempCP):

    # Padding patients to match the longest length.
    if patient.topColumns.shape[0] < maxPatientLen:

        fixedData = []
        for col in patient.topColumns:
            fixedData.append(np.pad(patient.topColumns[col], (0, maxPatientLen - patient.topColumns[col].shape[0]), 'c

        tempDF = pd.DataFrame(data = fixedData).T
        tempDF.columns = patient.topColumns.columns
        patient.topColumns = tempDF

    patientNonNullCount = patient.topColumns.count()


    # Interpolate with polynomial regression
    for column in patient.topColumns.columns:

        try:
            patient.interpolatedData[column] = patient.topColumns[column].interpolate(method='polynomial', order=2, li


        except ValueError:

            try:

                ## Use linear interpolation if polynomial fails
                if patientNonNullCount[column] == 1:
                    patient.interpolatedData[column] = patient.topColumns[column].interpolate(method='linear', limit_c
                else:
                    patient.interpolatedData[column] = patient.topColumns[column].interpolate(method='linear', limit_c

            except ValueError:

                patient.interpolatedData[column] = patient.topColumns[column].fillna(patient.topColumns[column].mean()
                noInterpolation += 1


print(f"{noInterpolation}/{len(clusteringPatients)} patients failed to interpolate")
```

In [ ]:

```
clusteringPatients = tempCP
```

In [ ]:

```
clusteringPatients[4].interpolatedData
```

Out[ ]:

| | PatientID | HR | O2Sat | Resp | MAP | SBP | Temp | Glucose | Potassium | BUN | Creatinine | Hct | Hgb | Platelets | WBC | Mag |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| **1344** | p000005 | 84.0 | 97.5 | 17.5 | 94.500000 | 140.5 | 37.280000 | NaN | NaN | NaN | NaN | NaN | NaN | NaN | NaN | |
| **1345** | p000005 | 80.0 | 99.0 | 18.0 | 99.000000 | 150.0 | 37.227660 | NaN | NaN | NaN | NaN | NaN | NaN | NaN | NaN | |
| **1346** | p000005 | 74.0 | 97.0 | 19.0 | 103.000000 | 142.0 | 37.220000 | NaN | NaN | NaN | NaN | NaN | NaN | NaN | NaN | |
| **1347** | p000005 | 73.0 | 98.0 | 17.0 | 99.000000 | 144.0 | 37.257019 | NaN | NaN | NaN | NaN | NaN | NaN | NaN | NaN | |
| **1348** | p000005 | 71.0 | 97.0 | 17.0 | 93.501172 | 144.0 | 37.338717 | 138.0 | 3.1 | 7.0 | 0.6 | 41.0 | 14.2 | 273.0 | 8.1 | |
| **...** | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... |
| **1675** | p000005 | NaN | NaN | NaN | NaN | NaN | NaN | NaN | NaN | NaN | NaN | NaN | NaN | NaN | NaN | |

| | PatientID | HR | O2Sat | Resp | MAP | SBP | Temp | Glucose | Potassium | BUN | Creatinine | Hct | Hgb | Platelets | WBC | Mag |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| **1676** | p000005 | NaN | NaN | NaN | NaN | NaN | NaN | NaN | NaN | NaN | NaN | NaN | NaN | NaN | NaN | |
| **1677** | p000005 | NaN | NaN | NaN | NaN | NaN | NaN | NaN | NaN | NaN | NaN | NaN | NaN | NaN | NaN | |
| **1678** | p000005 | NaN | NaN | NaN | NaN | NaN | NaN | NaN | NaN | NaN | NaN | NaN | NaN | NaN | NaN | |
| **1679** | p000005 | NaN | NaN | NaN | NaN | NaN | NaN | NaN | NaN | NaN | NaN | NaN | NaN | NaN | NaN | |

336 rows × 16 columns

### Checkpointing the processing here

In [ ]:
```
withIDs = []

for patient in clusteringPatients:
    tempP = copy.deepcopy(patient)
    tempP.interpolatedData['PatientID'] = tempP.patientID
    tempP.interpolatedData['Mortality14Days'] = tempP.label
    withIDs.append(tempP)


cleanedTimeSeriesDF = pd.concat([patient.interpolatedData for patient in withIDs])

cleanedTimeSeriesDF = cleanedTimeSeriesDF.set_index('PatientID')

cleanedTimeSeriesDF.to_csv("cleanedTemporalSepsisData.csv")
```

# Reload cached interpolated data from here

### Saves about 20 mins of processing

In [ ]:
```
# Reading the original dataframe
sepsisDF = pd.read_csv('../LEN_Test/data/sepsis_data.csv')

th = TH()

staticColumns = ["Age", "Gender", "Unit1", "Unit2", "HospAdmTime", "ICULOS"]

# Split the original dataframe into patients
patients = th.get_patients(sepsisDF, by="Patient_id", label="SepsisLabel", static = staticColumns)

patientsDict = {patient.patientID : patient for patient in patients}

# Read the cached preprocessing data
cleanedTimeSeriesDF = pd.read_csv("./processingCache/cleanedTemporalSepsisData.csv")


loadedIDs = set(cleanedTimeSeriesDF['PatientID'].unique())

# Selecting only the patients that were selected during feature selection
for id, group in tqdm(cleanedTimeSeriesDF.groupby("PatientID")):
    label = group.iloc[0]['Mortality14Days']
    group = group.drop(columns=['PatientID', 'Mortality14Days'], axis=1)
    patientsDict[id].interpolatedData = group

clusteringPatients = [patientsDict[id] for id in patientsDict if id in loadedIDs]
```

```
100%|██████████| 40336/40336 [01:28<00:00, 456.92it/s]
100%|██████████| 34028/34028 [00:20<00:00, 1682.80it/s]
```

### Sanity Check

In [ ]:
```
print(len(clusteringPatients))
```

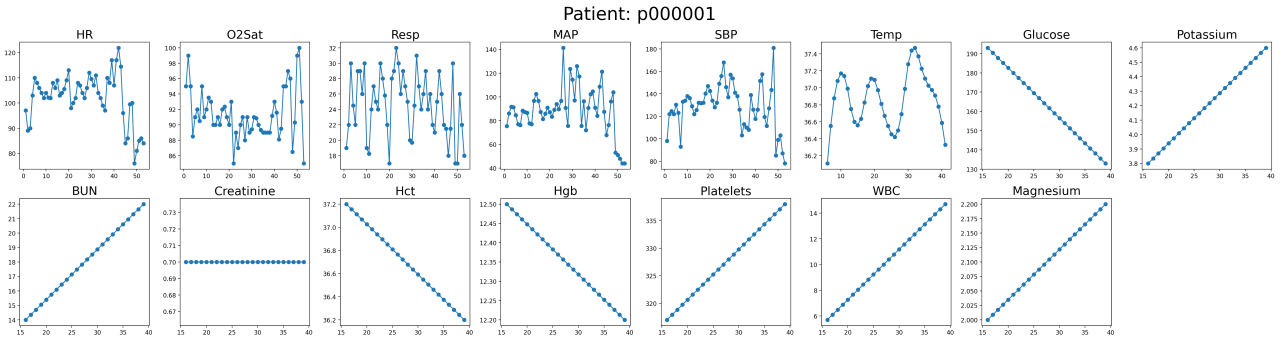
```
34028
```

In [ ]:
```
# The below graphing can only be run if caching has not been enabled.

# patient = clusteringPatients[1]

# fig = plt.figure(figsize = (30, 8), dpi=200)


# fig.suptitle(f"Patient: {patient.patientID}", fontsize=30)
```

```
# for idx, col in enumerate(patient.topColumns.columns):
#     plt.subplot(2, (len(patient.topColumns.columns)//2)+1, idx+1)

#     plt.scatter(patient.topColumns.index, patient.topColumns[col], c='Orange')
#     plt.title(f"{col}", fontsize=20)

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

In [ ]:

```
for patient in clusteringPatients[:1]:

    # display(clusteringPatients[i].interpolatedData.head())

    fig = plt.figure(figsize = (30, 8),dpi=200)


    fig.suptitle(f"Patient: {patient.patientID}", fontsize=30)

    for idx, col in enumerate(patient.interpolatedData.columns):
        plt.subplot(2, (len(patient.interpolatedData.columns)//2)+1, idx+1)

        plt.plot(patient.interpolatedData.index, patient.interpolatedData[col])
        plt.scatter(patient.interpolatedData.index, patient.interpolatedData[col])
        # plt.scatter(patient.topColumns.index, patient.topColumns[col], c="Orange")
        plt.title(f"{col}", fontsize=20)

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




In [ ]:

```
minorityClass = [patient for patient in clusteringPatients if patient.label == 1]
majorityClass = [patient for patient in clusteringPatients if patient.label == 0]

print(len(minorityClass))
print(len(majorityClass))
```

```
2422
31606
```

## Helper functions for clustering

In [ ]:

```
def formatForTimeSeries(column, sampleSize=None):

    # Getting an even split of target classes
    if sampleSize is None:
        sampleList = clusteringPatients
    else:
        minorityClass = [patient for patient in clusteringPatients if patient.label == 1]
        majorityClass = [patient for patient in clusteringPatients if patient.label == 0]

        minorityList = random.choices(minorityClass, k=sampleSize//2)
        majorityList = random.choices(majorityClass, k=sampleSize//2)

        sampleList = minorityList + majorityList



    print("Creating stacked DF...")
    stackedDF = pd.DataFrame([patient.interpolatedData[column].values for patient in sampleList])


    stackedNumpy = stackedDF.to_numpy()

    cleanedNumpy = []

    print("Cleaning")
    for row in stackedNumpy:
        cleanedNumpy.append(row[~np.isnan(row)])
```

```
    dataFormatted = to_time_series_dataset([*cleanedNumpy])

    return dataFormatted


def timeSeriesCluster(clusters, dataFormatted):

    print("Clustering")

    model = TimeSeriesKMeans(n_clusters=clusters, tol=1e-1, metric="dtw", max_iter=1, random_state=0, n_jobs=4)
    y_pred = model.fit_predict(dataFormatted)


    return y_pred, model
```

## Caching for clustering

In [ ]:

```
def find_cached(df=None, hash=None):

    if hash is None:

        print("Hashing...")
        hash = hashlib.sha256(bytes(str(df), 'utf-8')).hexdigest()

    display(hash)


    try:
        cachedDF = pd.read_csv("./processingCache/" + hash + ".csv").set_index("PatientID")

        print("Using cached df")

        return cachedDF, hash

    except:

        print("No cached df found")

        return False, hash
```

In [ ]:

```
myHash = "Chosen_clusters_sepsis"


clusteredDF, myHash = find_cached(clusteringPatients, hash=myHash)

if clusteredDF is False:

    clusteredDF = pd.DataFrame()

    for column in tqdm(clusteringPatients[0].interpolatedData.columns):
        dataFormatted = formatForTimeSeries(column, 1000)
        y_pred, model = timeSeriesCluster(2, dataFormatted)

        print("Finished fitting. Predicting... ")

        dataFormattedAll = formatForTimeSeries(column)
        y_pred = model.predict(dataFormattedAll)
        clusteredDF[column] = y_pred

    ids = [patient.patientID for patient in clusteringPatients]

    clusteredDF["PatientID"] = ids

    clusteredDF = clusteredDF.set_index("PatientID")


    clusteredDF.to_csv("./processingCache/" + myHash + ".csv")
```

```
'Chosen_clusters_sepsis'
Using cached df
```

# Reload clustered cached data from here

## Saves about 1.5 hours of processing

In [ ]:

```
# sepsisDF = pd.read_csv('../LEN_Test/data/sepsis_data.csv')

# th = TH()

# staticColumns = ["Age", "Gender", "Unit1", "Unit2", "HospAdmTime", "ICULOS"]

# patients = th.get_patients(sepsisDF, by="Patient_id", label="SepsisLabel", static = staticColumns)

# clusteredDF = pd.read_csv("./processingCache/Chosen_clusters_sepsis.csv").set_index("PatientID")

# clusteredDF = clusteredDF.set_index("PatientID")
```

In [ ]:

```
# fig = plt.figure(figsize = (15, 12), dpi=200)

# for idx, col in enumerate(colScores):
#     scores = [x[1] for x in colScores[col]]
#     plt.subplot(4, len(colScores)//4, idx+1)
#     plt.title(col)
#     plt.ylabel("Silhouette Score")
#     plt.xlabel("Num clusters")
#     plt.plot(list(range(2,2+len(scores))), scores)

# fig.suptitle(f"Silhouette scores for clusters 2 to {1+len(scores)}", fontsize=30)
# plt.tight_layout()
# plt.show()
```

In [ ]:

```
clusteredDF.describe()
```

Out[ ]:

| | HR | O2Sat | Resp | MAP | SBP | Temp | Glucose | Potassium | BUN | Cr |
|---|---|---|---|---|---|---|---|---|---|---|
| **count** | 34028.000000 | 34028.000000 | 34028.000000 | 34028.000000 | 34028.000000 | 34028.000000 | 34028.000000 | 34028.000000 | 34028.000000 | 3402 |
| **mean** | 0.423122 | 0.016751 | 0.468879 | 0.403286 | 0.460268 | 0.860027 | 0.000147 | 0.000059 | 0.116522 | |
| **std** | 0.494062 | 0.128339 | 0.499038 | 0.490564 | 0.498426 | 0.346964 | 0.012121 | 0.007666 | 0.320854 | |
| **min** | 0.000000 | 0.000000 | 0.000000 | 0.000000 | 0.000000 | 0.000000 | 0.000000 | 0.000000 | 0.000000 | |
| **25%** | 0.000000 | 0.000000 | 0.000000 | 0.000000 | 0.000000 | 1.000000 | 0.000000 | 0.000000 | 0.000000 | |
| **50%** | 0.000000 | 0.000000 | 0.000000 | 0.000000 | 0.000000 | 1.000000 | 0.000000 | 0.000000 | 0.000000 | |
| **75%** | 1.000000 | 0.000000 | 1.000000 | 1.000000 | 1.000000 | 1.000000 | 0.000000 | 0.000000 | 0.000000 | |
| **max** | 1.000000 | 1.000000 | 1.000000 | 1.000000 | 1.000000 | 1.000000 | 1.000000 | 1.000000 | 1.000000 | |

## Get an estimation of the true silhouette score with a sample of the entire dataset

In [ ]:

```
def silhouetteScoreCalc(data, y_pred, test_size=0.1):

    sample_idx = np.random.choice(data.shape[0], int(test_size * len(data)), replace=False)
    test_sample_x = data[sample_idx]
    test_sample_y = [y_pred[i] for i in sample_idx]

    patience = 0

    while len(np.unique(test_sample_y)) < 2:
        patience += 1
        if patience > 3:
            return 0
        print("Recalculating sample due to too few clusters")
        sample_idx = np.random.choice(data.shape[0], int(test_size * len(data)), replace=False)
        test_sample_x = data[sample_idx]
        test_sample_y = [y_pred[i] for i in sample_idx]

    score = silhouette_score(test_sample_x, test_sample_y, metric='dtw')
    print("Calculating sil score...")

    score = silhouette_score(test_sample_x, test_sample_y, metric='dtw', n_jobs=4)

    return score
```

In [ ]:

```
scores = {}

# Using sampling for the silhouette score since calculating the score on the entire dataset takes a long time
# The time series in this dataset are long so a smaller test size helps.
for column in tqdm(clusteredDF.columns):
    y_pred = list(clusteredDF[column])
    dataFormatted = formatForTimeSeries(column)
```

```
    score = silhouetteScoreCalc(dataFormatted, y_pred, test_size=0.001)


    scores[column] = score
```

## Remove outliers from the clustering graphs

In [ ]:

```
def removeOutliers(data, threshold):
    stdDev = np.nanstd(data)
    mean = np.nanmean(data)
    normalised = [np.nanmean(np.abs(d - mean)) for d in data]
    mask = normalised < threshold * stdDev
    return data[mask], data[np.logical_not(mask)]
```

## Creating the graphs takes a long time to run, commented out for speed

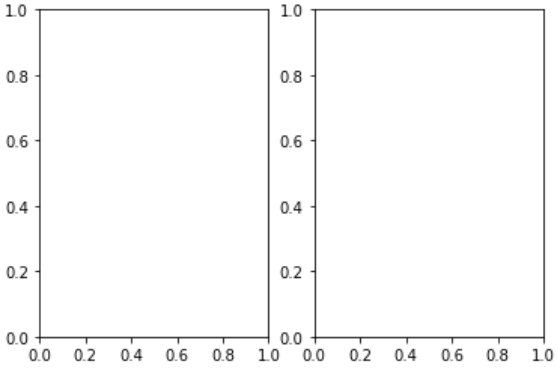
In [ ]:

```
colours = {0:'r', 1:'g', 2:'b', 3:'c', 4:'m', 5:'y', 6:'k', 7:'w', 8:'orange', 9:'purple', 10:'pink'}

clusterMetricsList = []

for col in clusteringPatients[0].interpolatedData.columns:

    clusters = 2

    # fig = plt.figure(figsize=(clusters*3,2.5), dpi=200)

    # fig.suptitle(f"{col}, Sil score: {np.round(scores[col], 2)}", fontsize=20)

    colData = [j.interpolatedData[col].values for j in clusteringPatients]

    minVal, maxVal = np.nanmin([np.nanmin(j) for j in colData]), np.nanmax([np.nanmax(j) for j in colData])

    formattedData = formatForTimeSeries(col)

    for i in range(clusters):
        # plt.subplot(1, clusters, i+1)

        y_pred = clusteredDF[col]

        # dataCluster = np.array(colData)[y_pred == i]

        dataCluster = formattedData[y_pred == i]

        withoutOutliers, outliers = removeOutliers(dataCluster, 1.5)

        # print(f"Num removed: {len(dataCluster) - len(withoutOutliers)}")

        # print(len(dataCluster))
        # print(len(withoutOutliers))

        # for sample in outliers:
        #     plt.plot(sample, c='black', alpha=0.05, linewidth=1)

        # for sample in withoutOutliers:
        #     plt.plot(sample, c=colours[i], alpha=0.1, linewidth=1)

        stdDev = np.nanstd(withoutOutliers)
        mean = np.nanmean(withoutOutliers)

        clusterMetricsList.append([col, stdDev, mean])

        # plt.title(f"C {i+1}, std: {np.round(stdDev, 2)}, mean: {np.round(mean, 2)}")
        # plt.xlabel("Time")
        # plt.ylabel("Value")
        # # print(dataCluster)
        # # print(np.nanstd(dataCluster))
        # plt.ylim(minVal, maxVal)


    # plt.tight_layout()
    # plt.savefig(f"./figures/sepsis/{col}.png")
    # plt.show()



clusterMetricsDF = pd.DataFrame(data = clusterMetricsList, columns=['Feature', 'StdDev', 'Mean'])
display(clusterMetricsDF)
```

```
Creating stacked DF...
Cleaning
```

```
Creating stacked DF...
Cleaning
Creating stacked DF...
Cleaning
Creating stacked DF...
Cleaning
Creating stacked DF...
Cleaning
Creating stacked DF...
Cleaning
Creating stacked DF...
Cleaning
Creating stacked DF...
Cleaning
Creating stacked DF...
Cleaning
Creating stacked DF...
Cleaning
Creating stacked DF...
Cleaning
Creating stacked DF...
Cleaning
Creating stacked DF...
Cleaning
Creating stacked DF...
Cleaning
Creating stacked DF...
Cleaning
Creating stacked DF...
Cleaning
```

| | Feature | StdDev | Mean |
|---|---|---|---|
| **0** | HR | 11.674412 | 75.263229 |
| **1** | HR | 12.965316 | 96.484716 |
| **2** | O2Sat | 2.550465 | 97.343287 |
| **3** | O2Sat | 6.912175 | 95.507201 |
| **4** | Resp | 3.879214 | 16.513264 |
| **5** | Resp | 4.890886 | 20.883793 |
| **6** | MAP | 11.410509 | 74.770268 |
| **7** | MAP | 14.671937 | 93.313586 |
| **8** | SBP | 15.426565 | 110.629313 |
| **9** | SBP | 20.635070 | 137.611479 |
| **10** | Temp | 0.660872 | 37.654162 |
| **11** | Temp | 0.592796 | 36.769462 |
| **12** | Glucose | 38.168913 | 127.698023 |
| **13** | Glucose | 1095.862999 | 925.709023 |
| **14** | Potassium | 0.514034 | 4.049991 |
| **15** | Potassium | 17.832761 | 1.746744 |
| **16** | BUN | 6.884820 | 15.957180 |
| **17** | BUN | 17.583774 | 57.217361 |
| **18** | Creatinine | 0.369612 | 0.935084 |
| **19** | Creatinine | 2.071211 | 5.785945 |
| **20** | Hct | 3.234219 | 36.577600 |
| **21** | Hct | 3.246277 | 28.558510 |
| **22** | Hgb | 1.103581 | 9.390612 |
| **23** | Hgb | 1.210000 | 12.114823 |
| **24** | Platelets | 61.937478 | 260.717022 |
| **25** | Platelets | 38.595953 | 137.349752 |
| **26** | WBC | 4.177366 | 14.661255 |
| **27** | WBC | 2.086208 | 8.036235 |
| **28** | Magnesium | 0.229804 | 1.980608 |
| **29** | Magnesium | 0.345381 | 2.506843 |




## Helper code to combine the above graphs into one image for the dissertation

In [ ]:

```
# figdir = "./figures/sepsis/"

# images = [Image.open(figdir + x) for x in list(next(os.walk(figdir))[2:])[0]]

# widths, heights = zip(*(i.size for i in images))

# widthMax = max(widths)
# widthMin = min(widths)
# heightTotal = sum(heights)

# combined = Image.new('RGBA', (widthMax, heightTotal))

# offset = 0
# for im in images:
#   xOffset = 0
#   if im.size[0] != widthMax:
#     xOffset = (widthMax - im.size[0]) // 2
#   combined.paste(im, (xOffset, offset))
#   offset += im.size[1]


# fig = plt.figure(figsize=(widthMax/100, heightTotal/100), dpi=100)
# plt.title("Result of Sepsis DTW clustering", fontsize=50)
# plt.axis('off')
# # plt.tight_layout()
# plt.imshow(combined)
# plt.show()


# # new_im.save('test.jpg')
```

## Order by std dev to find the clusters that vary the most, order by mean to find the highest/lowest values.

In [ ]:

```
display(clusteredDF.head())

def getMapping(metric, subset):

    ordered = subset.reset_index().sort_values(by=metric, ascending=True)

    before = ordered.index
    after = ordered.reset_index().index

    mapping = {before[i]: after[i] for i in range(len(before))}

    return mapping


orderedDF = pd.DataFrame()

for name, subset in clusterMetricsDF.groupby('Feature'):

    # clusterData = [np.pad(j.interpolatedData[col].values, (0, 48 - len(j.interpolatedData[col].values)), 'constant',

    for metric in list(clusterMetricsDF.columns)[1:]:

        mapping = getMapping(metric, subset)

        newCol = str(name + "_" + metric)

        orderedDF[newCol] = clusteredDF[name].map(mapping)
```

```
# orderedDF = orderedDF.set_index(clusteredDF.index)

display(orderedDF.head())
```

| PatientID | HR | O2Sat | Resp | MAP | SBP | Temp | Glucose | Potassium | BUN | Creatinine | Hct | Hgb | Platelets | WBC | Magnesium |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| **p000001** | 1 | 0 | 1 | 1 | 1 | 1 | 0 | 0 | 0 | 0 | 0 | 1 | 0 | 1 | 0 |
| **p000002** | 0 | 0 | 0 | 0 | 1 | 1 | 0 | 0 | 1 | 0 | 1 | 0 | 1 | 0 | 1 |
| **p000003** | 0 | 0 | 1 | 0 | 1 | 0 | 0 | 0 | 0 | 0 | 1 | 0 | 0 | 1 | 1 |
| **p000004** | 1 | 0 | 1 | 0 | 0 | 1 | 0 | 0 | 0 | 0 | 1 | 0 | 1 | 1 | 0 |
| **p000005** | 0 | 0 | 0 | 1 | 1 | 1 | 0 | 0 | 0 | 0 | 0 | 1 | 0 | 1 | 1 |

| PatientID | BUN_StdDev | BUN_Mean | Creatinine_StdDev | Creatinine_Mean | Glucose_StdDev | Glucose_Mean | HR_StdDev | HR_Mean | Hct_StdDev |
|---|---|---|---|---|---|---|---|---|---|
| **p000001** | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 1 | 0 |
| **p000002** | 1 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 1 |
| **p000003** | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 |
| **p000004** | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 1 | 1 |
| **p000005** | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

5 rows × 30 columns

In [ ]:
```
staticVals = [p.static.max().values for p in clusteringPatients]

staticDF = pd.DataFrame(data = staticVals, columns=staticColumns)

ids = [p.patientID for p in clusteringPatients]

staticDF['PatientID'] = ids

staticDF = staticDF.set_index("PatientID")

staticDF = staticDF.apply(lambda x: x.fillna(x.mean()))

staticDF
```

Out[ ]:

| PatientID | Age | Gender | Unit1 | Unit2 | HospAdmTime | ICULOS |
|---|---|---|---|---|---|---|
| **p000001** | 83.14 | 0.0 | 0.492377 | 0.507623 | -0.03 | 54.0 |
| **p000002** | 75.91 | 0.0 | 0.000000 | 1.000000 | -98.60 | 23.0 |
| **p000003** | 45.82 | 0.0 | 1.000000 | 0.000000 | -1195.71 | 48.0 |
| **p000004** | 65.71 | 0.0 | 0.000000 | 1.000000 | -8.77 | 29.0 |
| **p000005** | 28.09 | 1.0 | 1.000000 | 0.000000 | -0.05 | 49.0 |
| **...** | ... | ... | ... | ... | ... | ... |
| **p119995** | 76.00 | 1.0 | 0.000000 | 1.000000 | -14.90 | 42.0 |
| **p119996** | 84.00 | 0.0 | 0.492377 | 0.507623 | -6.69 | 48.0 |
| **p119997** | 30.00 | 1.0 | 0.492377 | 0.507623 | -0.02 | 25.0 |
| **p119998** | 60.00 | 0.0 | 1.000000 | 0.000000 | -53.64 | 49.0 |
| **p120000** | 62.00 | 0.0 | 0.492377 | 0.507623 | 0.00 | 35.0 |

34028 rows × 6 columns

In [ ]:
```
cat = Categorization.Categorizer(staticDF)

binnedDF = cat.kBins(2, 'uniform')

boundaries = cat.getBoundaries()

display(boundaries)
```

```
binnedDF['PatientID'] = ids

binnedDF = binnedDF.set_index("PatientID")

binnedDF = binnedDF.astype(np.int64)


binnedDF
```

```
{'kBins': {'Age': [57.0],
  'Gender': [1.0],
  'Unit1': [1.0],
  'Unit2': [0.507623403849819],
  'HospAdmTime': [-2562.53],
  'ICULOS': [172.0]}}
```

Out[ ]:

| PatientID | Age | Gender | Unit1 | Unit2 | HospAdmTime | ICULOS |
|---|---|---|---|---|---|---|
| **p000001** | 1 | 0 | 0 | 1 | 1 | 0 |
| **p000002** | 1 | 0 | 0 | 1 | 1 | 0 |
| **p000003** | 0 | 0 | 1 | 0 | 1 | 0 |
| **p000004** | 1 | 0 | 0 | 1 | 1 | 0 |
| **p000005** | 0 | 1 | 1 | 0 | 1 | 0 |
| **...** | ... | ... | ... | ... | ... | ... |
| **p119995** | 1 | 1 | 0 | 1 | 1 | 0 |
| **p119996** | 1 | 0 | 0 | 1 | 1 | 0 |
| **p119997** | 0 | 1 | 0 | 1 | 1 | 0 |
| **p119998** | 1 | 0 | 1 | 0 | 1 | 0 |
| **p120000** | 1 | 0 | 0 | 1 | 1 | 0 |

34028 rows × 6 columns

In [ ]:

```
orderedDF[staticColumns] = binnedDF[staticColumns]

orderedDF
```

Out[ ]:

| PatientID | BUN_StdDev | BUN_Mean | Creatinine_StdDev | Creatinine_Mean | Glucose_StdDev | Glucose_Mean | HR_StdDev | HR_Mean | Hct_StdDev |
|---|---|---|---|---|---|---|---|---|---|
| **p000001** | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 1 | 0 |
| **p000002** | 1 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 1 |
| **p000003** | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 |
| **p000004** | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 1 | 1 |
| **p000005** | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| **...** | ... | ... | ... | ... | ... | ... | ... | ... | ... |
| **p119995** | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| **p119996** | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 1 | 0 |
| **p119997** | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| **p119998** | 1 | 1 | 1 | 1 | 0 | 0 | 0 | 0 | 1 |
| **p120000** | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

34028 rows × 36 columns

In [ ]:

```
cat = Categorization.Categorizer()

mapping = {0: 'very_low', 1: 'low', 2: 'medium', 3: 'high', 4: 'very_high'}

mapped = cat.map_types(data = {"ordered":orderedDF}, mapping=mapping)['ordered']

display(mapped)
```

| PatientID | BUN_StdDev_high | BUN_StdDev_low | BUN_Mean_high | BUN_Mean_low | Creatinine_StdDev_high | Creatinine_StdDev_low | Creatinine_I |
|---|---|---|---|---|---|---|---|
| **p000001** | 0 | 1 | 0 | 1 | 0 | 1 | |
| **p000002** | 1 | 0 | 1 | 0 | 0 | 1 | |
| **p000003** | 0 | 1 | 0 | 1 | 0 | 1 | |
| **p000004** | 0 | 1 | 0 | 1 | 0 | 1 | |
| **p000005** | 0 | 1 | 0 | 1 | 0 | 1 | |
| **...** | ... | ... | ... | ... | ... | ... | |
| **p119995** | 0 | 1 | 0 | 1 | 0 | 1 | |
| **p119996** | 0 | 1 | 0 | 1 | 0 | 1 | |
| **p119997** | 0 | 1 | 0 | 1 | 0 | 1 | |
| **p119998** | 1 | 0 | 1 | 0 | 1 | 0 | |
| **p120000** | 0 | 1 | 0 | 1 | 0 | 1 | |

34028 rows × 72 columns

In [ ]:
```
targetSeries = [patient.label for patient in clusteringPatients]

# targetSeries
```

In [ ]:
```
mapped['Mortality14Days'] = targetSeries

display(mapped)

mapped.to_csv("./categorisedData/clusteredDataSepsis.csv")
```

| PatientID | BUN_StdDev_high | BUN_StdDev_low | BUN_Mean_high | BUN_Mean_low | Creatinine_StdDev_high | Creatinine_StdDev_low | Creatinine_I |
|---|---|---|---|---|---|---|---|
| **p000001** | 0 | 1 | 0 | 1 | 0 | 1 | |
| **p000002** | 1 | 0 | 1 | 0 | 0 | 1 | |
| **p000003** | 0 | 1 | 0 | 1 | 0 | 1 | |
| **p000004** | 0 | 1 | 0 | 1 | 0 | 1 | |
| **p000005** | 0 | 1 | 0 | 1 | 0 | 1 | |
| **...** | ... | ... | ... | ... | ... | ... | |
| **p119995** | 0 | 1 | 0 | 1 | 0 | 1 | |
| **p119996** | 0 | 1 | 0 | 1 | 0 | 1 | |
| **p119997** | 0 | 1 | 0 | 1 | 0 | 1 | |
| **p119998** | 1 | 0 | 1 | 0 | 1 | 0 | |
| **p120000** | 0 | 1 | 0 | 1 | 0 | 1 | |

34028 rows × 73 columns

In [ ]:
```
mapped['Mortality14Days'].value_counts()
```

Out[ ]:
```
0    31606
1     2422
Name: Mortality14Days, dtype: int64
```