

Mortality 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 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
import random
```

Loading in the MIMIC dataset

```
In [ ]: th = TH()
mimicDF = th.get_mimic()
mimicDF
```

	PatientID	Mortality14Days	ALT	AST	Admit Ht	Albumin	Arterial BP Mean	Arterial BP [Diastolic]	Arterial BP [Systolic]	Arterial PaCO2	...	SVI	SVR	SVRI	SaO2	Sodi
0	178177	0	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	...	NaN	NaN	NaN	NaN	NaN
1	178177	0	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	...	NaN	NaN	NaN	NaN	NaN
2	178177	0	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	...	NaN	NaN	NaN	NaN	NaN
3	178177	0	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	...	NaN	NaN	NaN	NaN	NaN
4	178177	0	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	...	NaN	NaN	NaN	NaN	NaN
...
47083	159740	0	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	...	NaN	NaN	NaN	NaN	NaN
47084	159740	0	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	...	NaN	NaN	NaN	NaN	NaN
47085	159740	0	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	...	NaN	NaN	NaN	NaN	NaN
47086	159740	0	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	...	NaN	NaN	NaN	NaN	NaN
47087	159740	0	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	...	NaN	NaN	NaN	NaN	NaN

47088 rows × 44 columns

Exploring values

```
In [ ]: mimicDF[mimicDF['Arterial pH'] > 8]
```

Out[]:

PatientID	Mortality14Days	ALT	AST	Admit Ht	Albumin	Arterial BP Mean	Arterial BP [Diastolic]	Arterial BP [Systolic]	Arterial PaCO2	...	SVI	SVR	SVRI	SaO2	Sodi
36229	198163	0	NaN	NaN	NaN	NaN	87.0	76.0	120.0	29.0	...	NaN	NaN	NaN	NaN

1 rows × 44 columns



```
In [ ]: mimicDF[~mimicDF['Admit Ht'].isnull()]
```

Out[]:

PatientID	Mortality14Days	ALT	AST	Admit Ht	Albumin	Arterial BP Mean	Arterial BP [Diastolic]	Arterial BP [Systolic]	Arterial PaCO2	...	SVI	SVR		
18	178177	0	NaN	NaN	65.0	NaN	86.0	74.0	122.0	NaN	...	19.178101	1644.439941	3609.7
54	110594	0	NaN	NaN	68.0	NaN	NaN	49.0	110.0	46.0	...	NaN	NaN	NaN
112	157699	0	NaN	NaN	64.0	NaN	NaN	NaN	NaN	42.0	...	NaN	NaN	NaN
142	157699	0	NaN	NaN	64.0	NaN	NaN	NaN	NaN	NaN	...	NaN	NaN	NaN
158	180230	0	NaN	NaN	70.0	NaN	NaN	NaN	NaN	38.0	...	NaN	NaN	NaN
...
46765	153586	0	NaN	NaN	62.0	NaN	NaN	NaN	NaN	38.0	...	NaN	NaN	NaN
46803	131057	0	NaN	NaN	73.0	NaN	NaN	NaN	NaN	NaN	...	NaN	NaN	NaN
46851	196595	0	NaN	NaN	70.0	NaN	NaN	NaN	NaN	NaN	...	NaN	NaN	NaN
46959	116726	0	NaN	NaN	62.0	NaN	55.0	40.0	90.0	NaN	...	NaN	1454.550049	2461.!
46970	169975	0	NaN	NaN	62.0	NaN	NaN	NaN	NaN	50.0	...	NaN	NaN	NaN

705 rows × 44 columns

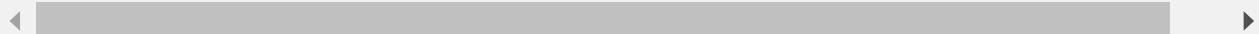


Describing all columns

```
In [ ]: # Too many columns to display all in one cell.
step = 10

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

	PatientID	Mortality14Days	ALT	AST	Admit Ht	Albumin	Arterial BP Mean	Arterial BP [Diastolic]	Arterial BP [Systolic]	A
count	47088.000000	47088.000000	484.000000	481.000000	705.000000	245.000000	31415.000000	31503.000000	31504.000000	8601.0
mean	151079.910805	0.039755	630.123967	954.301455	67.382553	2.721633	77.058157	57.226709	117.544502	40.!
std	29378.613191	0.195386	1245.805613	2384.326867	5.448469	0.584693	14.324177	11.197415	21.311674	7.0
min	100059.000000	0.000000	3.000000	3.000000	48.000000	1.400000	0.000000	0.000000	0.000000	15.0
25%	126241.000000	0.000000	25.000000	49.000000	64.000000	2.400000	68.000000	50.000000	103.000000	36.0
50%	151857.000000	0.000000	97.000000	137.000000	68.000000	2.700000	75.000000	56.000000	115.000000	40.0
75%	176484.000000	0.000000	554.750000	797.000000	70.000000	3.000000	84.000000	63.000000	129.000000	44.0
max	199998.000000	1.000000	8100.000000	23060.000000	160.000000	4.400000	287.000000	191.000000	255.000000	91.0



	Arterial PaO2	Arterial pH	BUN	CVP	CaO2	Chloride	Creatinine	Daily Weight	Fibrinogen	Glucose
count	8582.000000	9143.000000	1978.000000	27657.000000	3667.000000	1741.000000	1982.000000	956.000000	572.000000	7978.000000
mean	152.720982	7.400322	24.094034	11.796478	13.867385	107.547961	1.475732	91.145711	248.688986	125.757458
std	92.767596	1.879658	14.906156	5.155437	1.898516	4.878126	1.330165	22.804774	123.542862	40.834841
min	7.470000	6.920000	3.000000	0.000000	0.706800	87.000000	0.200000	0.000000	68.000000	26.000000

	Arterial PaO2	Arterial pH	BUN	CVP	CaO2	Chloride	Creatinine	Daily Weight	Fibrinogen	Glucose
25%	90.000000	7.340000	14.000000	8.000000	12.651100	105.000000	0.800000	75.500000	166.750000	101.000000
50%	118.000000	7.380000	20.000000	11.000000	13.784700	108.000000	1.100000	88.650002	213.500000	119.000000
75%	174.750000	7.420000	31.000000	15.000000	14.918100	111.000000	1.500000	104.099998	301.250000	142.000000
max	499.000000	187.000000	143.000000	50.000000	22.196800	133.000000	12.500000	208.199997	925.000000	638.000000

	Heart Rate	Hemoglobin	INR	Ionized Calcium	LDH	Magnesium	NBP Mean	NBP [Diastolic]	NBP [Systolic]	PTT
count	34081.000000	3487.000000	1909.000000	6518.000000	215.000000	1881.000000	5404.000000	5451.000000	5473.000000	1937.000000
mean	86.659987	10.381594	1.459246	1.247281	1209.302326	2.138437	73.803848	54.465603	114.147040	41.343211
std	14.619726	1.536069	0.857355	3.527599	1870.129434	0.417471	13.540250	13.884506	21.738575	20.785616
min	0.000000	0.000000	0.600000	0.490000	152.000000	0.800000	0.000000	0.000000	0.000000	21.300000
25%	78.000000	9.400000	1.200000	1.090000	318.500000	1.900000	64.666702	45.000000	100.000000	30.400000
50%	86.000000	10.300000	1.300000	1.150000	439.000000	2.100000	72.333298	53.000000	112.000000	35.400000
75%	95.000000	11.200000	1.500000	1.200000	1037.000000	2.300000	81.333298	63.000000	126.000000	43.600000
max	223.000000	19.100000	27.000000	160.000000	12240.000000	4.500000	149.000000	125.000000	234.000000	150.000000

	Platelets	Potassium	Resp Rate (Spont)	Respiratory Rate (spontaneous)	SVI	SVR	SVRI	SaO2	Sodium		
count	3222.000000	5717.000000	4043.000000		1.0	12981.000000	14768.000000	14714.000000	4352.000000	2485.000000	32961.000000
mean	142.337678	15.009138	2.345041		5.0	33.217492	987.290546	1917.421073	96.799012	138.022942	97.210000
std	65.006060	810.131474	4.719276		NaN	10.474703	363.504762	608.801780	2.659592	5.435978	3.010000
min	18.000000	2.300000	0.000000		5.0	0.000000	0.000000	0.000000	42.000000	1.210000	0.010000
25%	97.000000	3.900000	0.000000		5.0	26.168200	737.500000	1494.877533	96.000000	136.000000	96.010000
50%	131.000000	4.200000	0.000000		5.0	31.460699	931.028503	1841.049988	97.000000	138.000000	98.010000
75%	174.000000	4.600000	3.000000		5.0	38.118801	1169.229980	2244.340088	98.000000	140.000000	100.010000
max	689.000000	61259.000000	38.000000		5.0	179.212997	3784.620117	6000.000000	100.000000	164.000000	100.010000

	SvO2	Temperature C	Total Bili	WBC
count	85.000000	26516.000000	331.000000	2118.000000
mean	64.988235	37.235334	2.529305	12.747691
std	10.518118	1.399548	3.004787	5.523420
min	43.000000	0.000000	0.000000	1.900000
25%	58.000000	36.900002	0.600000	9.000000
50%	65.000000	37.299999	1.400000	11.900000
75%	71.000000	37.700001	3.300000	15.500000
max	100.000000	46.500000	16.400000	48.300000

```
In [ ]: # Fixing 'arterial pH', 'ionized calcium' since they contain erroneous values.
```

```
mimicDF['Arterial pH'][mimicDF['Arterial pH'] > 8] = mimicDF['Arterial pH'].mean()
mimicDF['Ionized Calcium'][mimicDF['Ionized Calcium'] > 8] = mimicDF['Ionized Calcium'].mean()
```

```
C:\Users\benma\AppData\Local\Temp\ipykernel_10712\153206369.py:3: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame
```

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

```
mimicDF['Arterial pH'][mimicDF['Arterial pH'] > 8] = mimicDF['Arterial pH'].mean()
```

```
C:\Users\benma\AppData\Local\Temp\ipykernel_10712\153206369.py:4: SettingWithCopyWarning:
```

```
A value is trying to be set on a copy of a slice from a DataFrame
```

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

```
mimicDF['Ionized Calcium'][mimicDF['Ionized Calcium'] > 8] = mimicDF['Ionized Calcium'].mean()
```

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

There are 1126 unique patients in the dataset

```
In [ ]: patients = th.get_patients(mimicDF)

print(len(patients))
```

100%|██████████| 1126/1126 [00:01<00:00, 721.32it/s]
1126

Counting total null columns across all patients

```
In [ ]: totalNullColumns = 0

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

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

print(totalColumns, totalNullColumns)

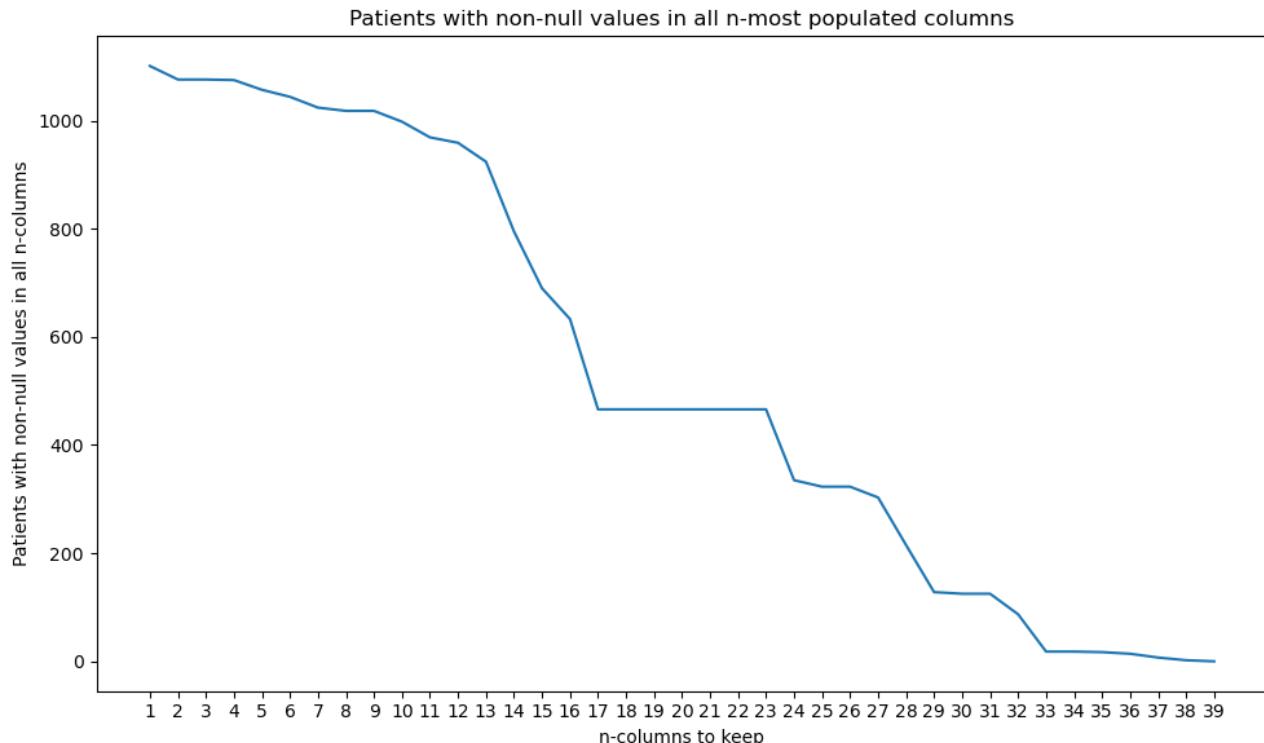
print(f"{np.round(totalNullColumns / totalColumns * 100, 2)}% of columns are null")
```

47292 15553
32.89% of columns are null

Counting how many columns have no data in

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

[1/4] Counting null values...
100%|██████████| 1126/1126 [00:00<00:00, 1455.21it/s]
[2/3] Copying dataset...
[3/4] Dropping null columns...
100%|██████████| 39/39 [00:03<00:00, 12.46it/s]
[4/4] Graphing...
```



Sharp drop off after 12 columns so will keep around 1000 patients with at least some data in the top 12 columns

```
In [ ]: clusteringPatients = th.get_top_columns(patients, 12)

# clusteringPatients = patients

# for patient in clusteringPatients:
#     patient.topColumns = patient.data
```

```
print(len(clusteringPatients))

clusteringPatients[0].topColumns
```

100%|██████████| 1126/1126 [00:00<00:00, 2414.22it/s]

959

Out[]:	Platelets	Arterial BP [Diastolic]	Arterial BP [Systolic]	Arterial BP Mean	CVP	Arterial pH	Hemoglobin	Arterial PaCO2	Arterial PaO2	SVR	Ionized Calcium	SVRI
0	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN
1	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN
2	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN
3	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN
4	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN
5	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN
6	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN
7	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN
8	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN
9	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN
10	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN
11	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN
12	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN
13	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN
14	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN
15	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN
16	NaN	NaN	NaN	NaN	7.33	NaN	53.0	437.0	NaN	1.19	NaN	NaN
17	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN
18	NaN	NaN	NaN	NaN	7.45	NaN	39.0	288.0	NaN	NaN	NaN	NaN
19	NaN	NaN	NaN	NaN	7.39	NaN	40.0	431.0	NaN	NaN	NaN	NaN
20	129.0	NaN	NaN	NaN	7.34	NaN	46.0	283.0	NaN	1.12	NaN	NaN
21	107.0	43.0	108.0	61.0	12.0	7.36	NaN	42.0	265.0	890.755981	1.27	1819.739990
22	NaN	47.0	136.0	72.0	11.0	7.34	NaN	41.0	108.0	1060.040039	NaN	2160.340088
23	NaN	51.0	123.0	69.0	10.0	NaN	NaN	NaN	1209.760010	NaN	2467.659912	NaN
24	NaN	51.0	114.0	67.0	8.0	7.29	NaN	42.0	114.0	1151.219971	0.94	2348.260010
25	NaN	61.0	132.0	77.0	10.0	7.34	NaN	45.0	150.0	1001.929993	1.15	2078.739990
26	150.0	54.0	125.0	102.0	10.0	7.32	NaN	50.0	83.0	NaN	1.29	NaN
27	NaN	53.0	120.0	73.0	9.0	7.34	NaN	41.0	95.0	NaN	1.05	NaN
28	NaN	54.0	133.0	77.0	15.0	7.32	NaN	45.0	97.0	997.987976	NaN	2032.790039
29	NaN	53.0	128.0	72.0	12.0	NaN	NaN	NaN	NaN	NaN	NaN	NaN
30	149.0	47.0	118.0	66.0	9.0	7.32	11.4	46.0	94.0	NaN	1.15	NaN
31	NaN	50.0	128.0	72.0	10.0	NaN	NaN	NaN	934.086975	NaN	1907.689941	NaN
32	NaN	61.0	150.0	86.0	14.0	NaN	NaN	NaN	NaN	NaN	NaN	NaN
33	NaN	49.0	132.0	75.0	11.0	NaN	NaN	NaN	NaN	NaN	NaN	NaN
34	NaN	58.0	147.0	83.0	18.0	7.33	NaN	43.0	120.0	NaN	1.16	NaN
35	NaN	44.0	110.0	60.0	15.0	NaN	NaN	NaN	NaN	NaN	NaN	NaN
36	NaN	46.0	107.0	62.0	15.0	NaN	NaN	NaN	NaN	NaN	NaN	NaN
37	NaN	52.0	121.0	71.0	14.0	NaN	NaN	NaN	NaN	NaN	NaN	NaN
38	NaN	45.0	113.0	64.0	16.0	NaN	NaN	NaN	NaN	NaN	NaN	NaN
39	162.0	44.0	109.0	60.0	16.0	7.31	11.7	45.0	98.0	739.495972	1.22	1510.729980
40	NaN	47.0	113.0	64.0	NaN	7.29	NaN	NaN	NaN	NaN	1.17	NaN
41	NaN	44.0	106.0	62.0	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN
42	NaN	61.0	133.0	80.0	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN

	Platelets	Arterial BP [Diastolic]	Arterial BP [Systolic]	Arterial BP Mean	CVP	Arterial pH	Hemoglobin	Arterial PaCO2	Arterial PaO2	SVR	Ionized Calcium	SVRI
43	NaN	62.0	133.0	81.0	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN
44	NaN	64.0	135.0	84.0	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN
45	NaN	60.0	139.0	81.0	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN
46	NaN	66.0	145.0	92.0	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN
47	NaN	52.0	132.0	75.0	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN

```
In [ ]: allDataTemp = pd.concat([patient.data for patient in clusteringPatients])
allDataTemp = allDataTemp.describe().T['mean'].T
allDataTemp['ALT']
```

Out[]: 672.8090692124105

```
In [ ]: clusteringPatients[0].topColumns.count()
```

```
Out[ ]: Platelets      5
Arterial BP [Diastolic] 27
Arterial BP [Systolic] 27
Arterial BP Mean 27
CVP          19
Arterial pH     15
Hemoglobin    2
Arterial PaCO2 14
Arterial PaO2 14
SVR          8
Ionized Calcium 11
SVRI         8
dtype: int64
```

Interpolating missing data

```
In [ ]: noInterpolation = 0
failureExample = (0,0)

for idx, patient in tqdm(enumerate(clusteringPatients)):

    patient.interpolatedData = copy.copy(patient.topColumns)

    patientNonNullCount = patient.topColumns.count()

    for column in patient.topColumns.columns:

        # If the column has no data in then fill with the mean from the other patients
        if patient.topColumns[column].isnull().all():
            print("replacing ", column)
            print(allDataTemp[column])
            patient.interpolatedData[column] = allDataTemp[column]
            continue

        try:
            # Try interpolating with a polynomial model
            patient.interpolatedData[column] = patient.topColumns[column].interpolate(method='polynomial', order=2, limit_direction='both')

        except ValueError:
            try:
                if patientNonNullCount[column] == 1:
                    # If the column has only one data point in, pad with three values either side.
                    patient.interpolatedData[column] = patient.topColumns[column].interpolate(method='linear', limit_direction='both')
                elif patientNonNullCount[column] == 0:
                    print("no data in ", column)
                    break
                else:
                    # Otherwise, interpolate with a Linear model
                    patient.interpolatedData[column] = patient.topColumns[column].interpolate(method='linear', limit_direction='both')

            except ValueError:
                # If the interpolation fails, then fill with the mean from the other patients
                patient.interpolatedData[column] = patient.topColumns[column].fillna(allDataTemp[column])
                noInterpolation += 1
```

```

failureExample = (idx, column)

if patient.interpolatedData.shape[0] != 48:
    # If hte column Length is shorter than 48, pad with nan values.
    fixedData = []
    for col in patient.interpolatedData:
        fixedData.append(np.pad(patient.interpolatedData[col], (0, 48 - patient.interpolatedData[col].shape[0]),

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

print(f"{noInterpolation}/{len(clusteringPatients)} patients failed to interpolate all columns")
print(f"{failureExample}")

```

959it [00:12, 75.60it/s]
0/959 patients failed to interpolate all columns
(0, 0)

In []:

```

for patient in clusteringPatients[:5]:
    display(patient.interpolatedData.head())

```

	Platelets	Arterial BP [Diastolic]	Arterial BP [Systolic]	Arterial BP Mean	CVP	Arterial pH	Hemoglobin	Arterial PaCO2	Arterial PaO2	SVR	Ionized Calcium	SVRI
0	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN
1	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN
2	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN
3	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN
4	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN
	Platelets	Arterial BP [Diastolic]	Arterial BP [Systolic]	Arterial BP Mean	CVP	Arterial pH	Hemoglobin	Arterial PaCO2	Arterial PaO2	SVR	Ionized Calcium	SVRI
0	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN
1	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN
2	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN
3	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN
4	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN
	Platelets	Arterial BP [Diastolic]	Arterial BP [Systolic]	Arterial BP Mean	CVP	Arterial pH	Hemoglobin	Arterial PaCO2	Arterial PaO2	SVR	Ionized Calcium	SVRI
0	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN
1	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN
2	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN
3	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN
4	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN
	Platelets	Arterial BP [Diastolic]	Arterial BP [Systolic]	Arterial BP Mean	CVP	Arterial pH	Hemoglobin	Arterial PaCO2	Arterial PaO2	SVR	Ionized Calcium	SVRI
0	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN
1	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN
2	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN
3	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN
4	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN
	Platelets	Arterial BP [Diastolic]	Arterial BP [Systolic]	Arterial BP Mean	CVP	Arterial pH	Hemoglobin	Arterial PaCO2	Arterial PaO2	SVR	Ionized Calcium	SVRI
0	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN
1	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN
2	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN
3	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN
4	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN
	Platelets	Arterial BP [Diastolic]	Arterial BP [Systolic]	Arterial BP Mean	CVP	Arterial pH	Hemoglobin	Arterial PaCO2	Arterial PaO2	SVR	Ionized Calcium	SVRI
0	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN
1	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN
2	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN
3	149.0	NaN	NaN	NaN	NaN	7.31	10.3	49.0	147.0	NaN	1.33	NaN

	Platelets	Arterial BP [Diastolic]	Arterial BP [Systolic]	Arterial BP Mean	CVP	Arterial pH	Hemoglobin	Arterial PaCO2	Arterial PaO2	SVR	Ionized Calcium	SVRI
4	154.0	75.0	111.0	90.0	10.0	7.27	12.1	58.0	375.0	998.440002	1.34	2184.300049

Visualising the data before interpolation

```
In [ ]: patient = clusteringPatients[0]

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

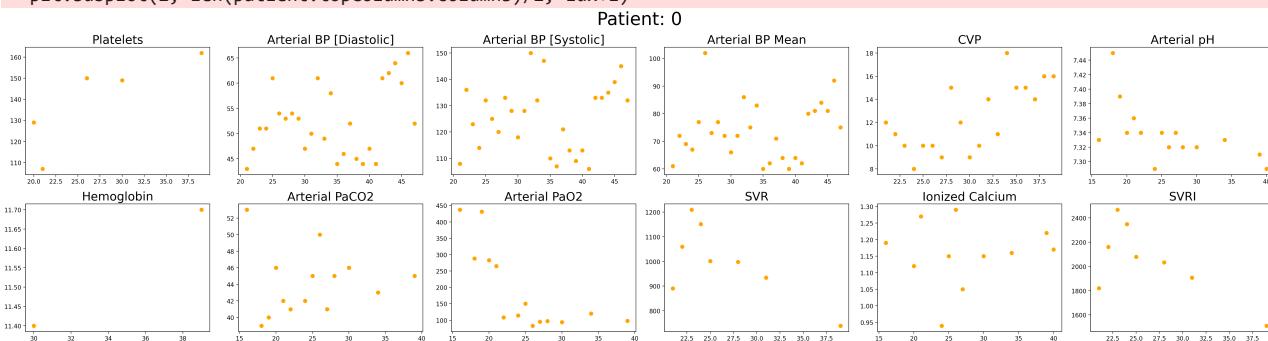
fig.suptitle(f"Patient: {patient.label}", fontsize=30)

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

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

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

C:\Users\benma\AppData\Local\Temp\ipykernel_10712/2467793513.py:9: MatplotlibDeprecationWarning: Passing non-integers as three-element position specification is deprecated since 3.3 and will be removed two minor releases later.
plt.subplot(2, len(patient.topColumns.columns)/2, idx+1)



Visualising the data after interpolation

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

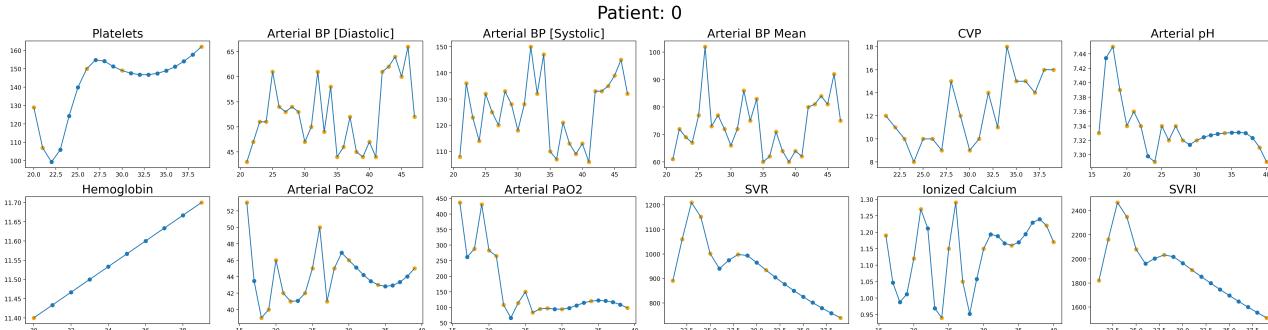
    fig = plt.figure(figsize = (30, 8),dpi=200)

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

    for idx, col in enumerate(patient.interpolatedData.columns):
        plt.subplot(2, len(patient.interpolatedData.columns)//2, 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 [ ]: # Format the given column for clustering
def formatForTimeSeries(column, sampleSize=None):

    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(minorityList)

    # sampleList = random.shuffle(sampleList)

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:
    cRow = row[~np.isnan(row)]
    if len(cRow) > 0:
        cleanedNumpy.append(cRow)

dataFormatted = to_time_series_dataset(*cleanedNumpy)

return dataFormatted

# Cluster the given column
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)
    model.fit(dataFormatted)

    return model

```

Establishing the caching function

```
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
```

Clustering all columns

```
In [ ]: myHash = "Chosen_clusters"

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

#Originally each column was a separate cluster, but this was changed to 2 for all since it gave best performance.
```

```

clusterNums = {"Platelets": 3, "Arterial BP [Diastolic)": 2, "Arterial BP [Systolic)": 2, "Arterial BP Mean": 2, "CVP": 1, "Arterial pH": 1, "Hemoglobin": 1, "Arterial PaCO2": 1, "Arterial PaO2": 1, "SVR": 1, "Ionized Calcium": 1, "SVRI": 1}

# Caching disabled
if clusteredDF is False:

    clusteredDF = pd.DataFrame()

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

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

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

    ids = [patient.label for patient in clusteringPatients]

    clusteredDF["PatientID"] = ids

    clusteredDF = clusteredDF.set_index("PatientID")

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

```

'Chosen_clusters'
Using cached df

In []: clusteringPatients[2].interpolatedData

	Platelets	Arterial BP [Diastolic]	Arterial BP [Systolic]	Arterial BP Mean	CVP	Arterial pH	Hemoglobin	Arterial PaCO2	Arterial PaO2	SVR	Ionized Calcium	SVRI
0	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN
1	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN
2	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN
3	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN
4	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN
5	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN
6	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN
7	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN
8	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN
9	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN
10	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN
11	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN
12	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN
13	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN
14	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN
15	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN
16	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN
17	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN
18	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN
19	NaN	NaN	NaN	NaN	NaN	7.460000	NaN	36.000000	470.000000	NaN	NaN	NaN
20	NaN	NaN	NaN	NaN	NaN	7.430000	NaN	32.000000	325.000000	NaN	1.180000	NaN
21	87.0000	54.0	117.0	83.0	8.0	7.410000	NaN	32.000000	402.000000	1696.199951	1.210000	2791.669922
22	86.9375	77.0	140.0	96.0	9.0	7.391521	NaN	32.157869	372.827331	2035.089966	1.214756	3362.320068
23	86.8750	55.0	116.0	78.0	2.0	7.372607	NaN	32.274790	215.022365	1940.256229	1.194269	2994.398132
24	86.8125	60.0	86.0	61.0	12.0	7.350000	NaN	35.000000	122.000000	1682.400024	1.148538	2364.530029
25	86.7500	57.0	121.0	86.0	12.0	7.323157	NaN	40.775037	125.996053	1532.222637	1.082193	2149.341674
26	86.6875	54.0	95.0	71.0	13.0	7.310000	NaN	45.000000	166.000000	1478.573594	1.023007	2286.628146

	Platelets	Arterial BP [Diastolic]	Arterial BP [Systolic]	Arterial BP Mean	CVP	Arterial pH	Hemoglobin	Arterial PaCO2	Arterial PaO2	SVR	Ionized Calcium	SVRI
27	86.6250	59.0	117.0	80.0	14.0	7.328453	NaN	43.074988	181.001316	1454.550049	0.975609	2403.159912
28	86.5625	49.0	122.0	81.0	13.0	7.360000	NaN	35.000000	171.000000	1449.001528	0.940000	2436.732050
29	86.5000	52.0	122.0	74.0	9.0	7.386127	NaN	NaN	NaN	1390.369995	0.916179	2325.989990
30	86.4375	60.0	148.0	88.0	15.0	7.406833	NaN	NaN	NaN	1207.097413	0.904147	2009.579162
31	86.3750	50.0	117.0	70.0	11.0	7.422118	NaN	NaN	NaN	946.312993	0.903904	1566.411651
32	86.3125	49.0	130.0	78.0	26.0	7.431983	NaN	NaN	NaN	890.791992	0.915449	1469.959961
33	86.2500	50.0	126.0	71.0	11.0	7.436428	NaN	NaN	NaN	1041.099976	0.938782	1720.750000
34	86.1875	50.0	114.0	69.0	11.0	7.435452	7.1	NaN	NaN	1117.855070	0.973904	1848.464715
35	86.1250	50.0	108.0	66.0	10.0	7.429055	7.1	NaN	NaN	1074.493629	1.020814	1774.717930
36	86.0625	48.0	119.0	68.0	11.0	7.417238	7.1	NaN	NaN	987.991117	1.079513	1629.044675
37	86.0000	41.0	103.0	59.0	12.0	7.400000	7.1	NaN	NaN	935.322998	1.150000	1540.979980
38	NaN	52.0	118.0	73.0	18.0	NaN	7.1	NaN	NaN	916.489272	NaN	1510.523845
39	NaN	43.0	100.0	60.0	11.0	NaN	7.1	NaN	NaN	930.596163	NaN	1535.808798
40	NaN	54.0	120.0	69.0	12.0	NaN	7.1	NaN	NaN	972.281006	NaN	1605.630005
41	NaN	55.0	136.0	78.0	14.0	NaN	NaN	NaN	NaN	1040.650024	NaN	1718.119995
42	NaN	52.0	128.0	74.0	12.0	NaN	NaN	NaN	NaN	NaN	NaN	NaN
43	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN
44	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN
45	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN
46	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN
47	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN

In []: clusteredDF.describe()

Out[]:

	Platelets	Arterial BP [Diastolic]	Arterial BP [Systolic]	Arterial BP Mean	CVP	Arterial pH	Hemoglobin	Arterial PaCO2	Arterial PaO2	SVR	Ionized Calcium
count	959.000000	959.000000	959.000000	959.000000	959.000000	959.000000	959.000000	959.000000	959.000000	959.000000	959.000000
mean	0.704901	0.636079	0.289885	0.080292	0.570386	0.593326	0.600626	0.896767	0.377477	0.642336	0.847758
std	0.456325	0.481377	0.453946	0.271886	0.495279	0.491469	0.490025	0.304421	0.485009	0.479563	0.359443
min	0.000000	0.000000	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	0.000000	0.000000	1.000000	0.000000	0.000000	1.000000
50%	1.000000	1.000000	0.000000	0.000000	1.000000	1.000000	1.000000	1.000000	0.000000	1.000000	1.000000
75%	1.000000	1.000000	1.000000	0.000000	1.000000	1.000000	1.000000	1.000000	1.000000	1.000000	1.000000
max	1.000000	1.000000	1.000000	1.000000	1.000000	1.000000	1.000000	1.000000	1.000000	1.000000	1.000000

Custom random sampling silhouette function

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')

return score
```

In []:

```
scores = {}

# Using sampling for the silhouette score since calculating the score on the entire dataset takes a long time
for column in tqdm(clusteredDF.columns):
    y_pred = list(clusteredDF[column])
    dataFormatted = formatForTimeSeries(column)

    score = silhouetteScoreCalc(dataFormatted, y_pred)
    scores[column] = score

print(scores)
```

0% | 0/12 [00:00<?, ?it/s]
 Creating stacked DF...
 Cleaning
 8%|█ | 1/12 [00:02<00:26, 2.40s/it]
 Creating stacked DF...
 Cleaning
 17%|██ | 2/12 [00:03<00:16, 1.67s/it]
 Creating stacked DF...
 Cleaning
 25%|███ | 3/12 [00:04<00:12, 1.40s/it]
 Creating stacked DF...
 Cleaning
 33%|███ | 4/12 [00:05<00:10, 1.32s/it]
 Creating stacked DF...
 Cleaning
 42%|███ | 5/12 [00:07<00:10, 1.50s/it]
 Creating stacked DF...
 Cleaning
 50%|████ | 6/12 [00:09<00:09, 1.60s/it]
 Creating stacked DF...
 Cleaning
 58%|█████ | 7/12 [00:11<00:08, 1.65s/it]
 Creating stacked DF...
 Cleaning
 67%|█████ | 8/12 [00:13<00:07, 1.79s/it]
 Creating stacked DF...
 Cleaning
 75%|█████ | 9/12 [00:14<00:05, 1.72s/it]
 Creating stacked DF...
 Cleaning
 83%|█████ | 10/12 [00:16<00:03, 1.72s/it]
 Creating stacked DF...
 Cleaning
 92%|█████ | 11/12 [00:18<00:01, 1.78s/it]
 Creating stacked DF...
 Cleaning
 100%|██████ | 12/12 [00:20<00:00, 1.70s/it]
 {'platelets': 0.48820540856507627, 'Arterial BP [Diastolic]': 0.307493187139085, 'Arterial BP [Systolic]': 0.26802183239864935, 'Arterial BP Mean': 0.4898314225253402, 'CVP': 0.09713865063460426, 'Arterial pH': 0.2330295399640199, 'Hemo globin': 0.38581754693123793, 'Arterial PaCO2': 0.45736310434927996, 'Arterial PaO2': 0.3878189576209379, 'SVR': 0.37421003808938585, 'Ionized Calcium': 0.3880833177487159, 'SVRI': 0.3857458826255939}

Removing outliers from the clustering

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)]
```

Plotting the results of clustering

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 = [np.pad(j.interpolatedData[col].values, (0, 48 - len(j.interpolatedData[col].values)), 'constant', constant_value=np.nanmin([np.nanmin(j) for j in colData])), np.nanmax([np.nanmax(j) for j in colData])]

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

        y_pred = clusteredDF[col]

        dataCluster = formatForTimeSeries(col)[y_pred == i]

        withoutOutliers, outliers = removeOutliers(dataCluster, 1.5)

        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")
        plt.ylim(minVal, maxVal)

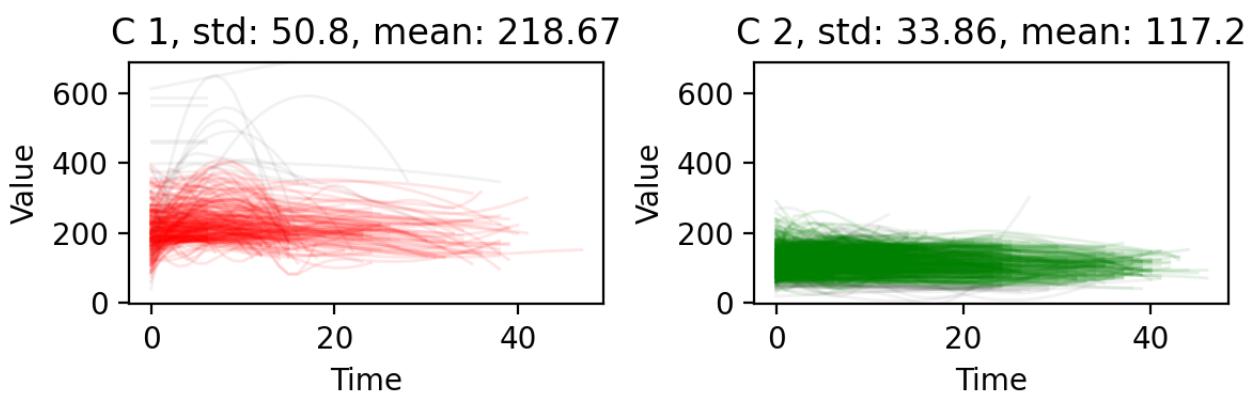
    plt.tight_layout()
    plt.savefig(f'./figures/{col}.png')
    plt.show()

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

```

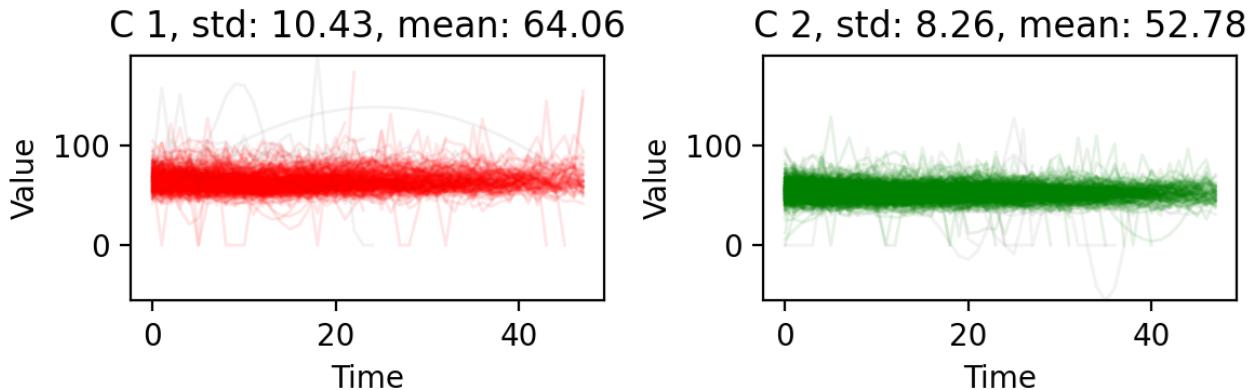
Creating stacked DF...
Cleaning
Creating stacked DF...
Cleaning

Platelets, Sil score: 0.49



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

Arterial BP [Diastolic], Sil score: 0.31



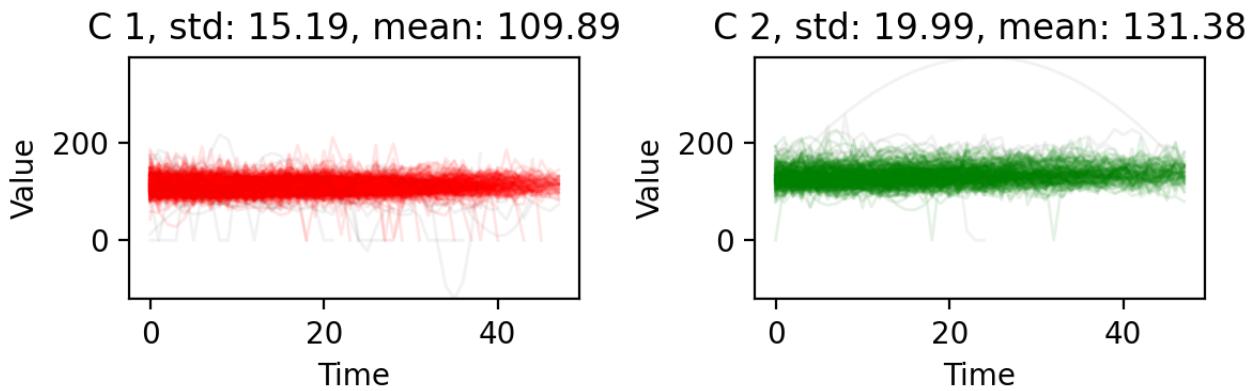
Creating stacked DF...

Cleaning

Creating stacked DF...

Cleaning

Arterial BP [Systolic], Sil score: 0.27



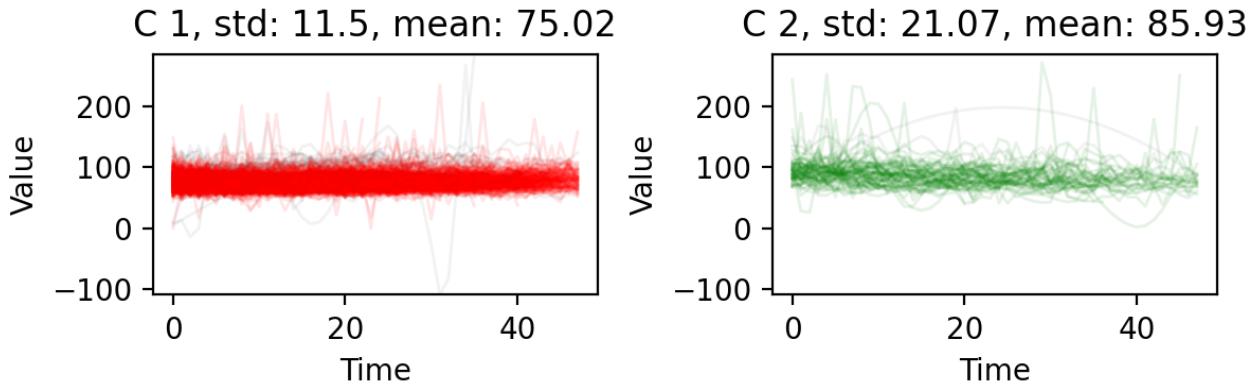
Creating stacked DF...

Cleaning

Creating stacked DF...

Cleaning

Arterial BP Mean, Sil score: 0.49



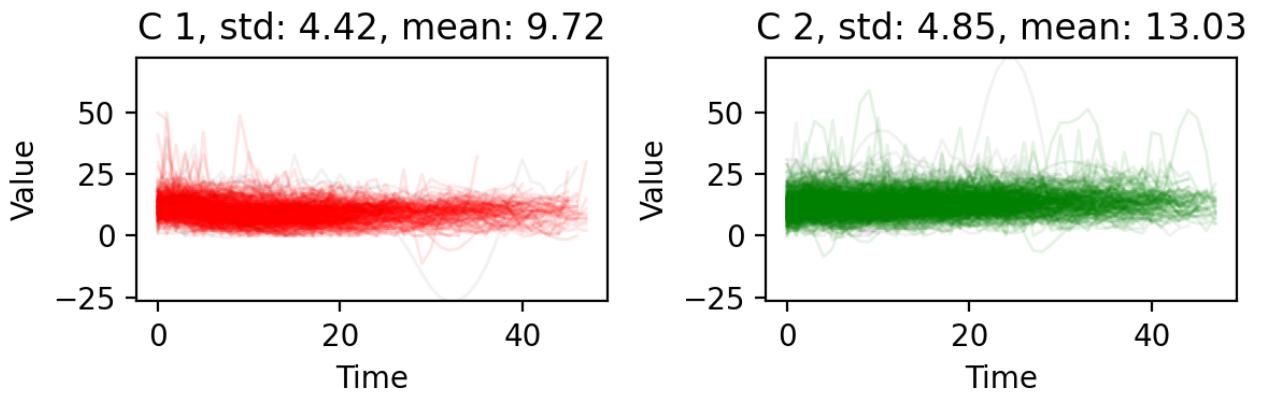
Creating stacked DF...

Cleaning

Creating stacked DF...

Cleaning

CVP, Sil score: 0.1



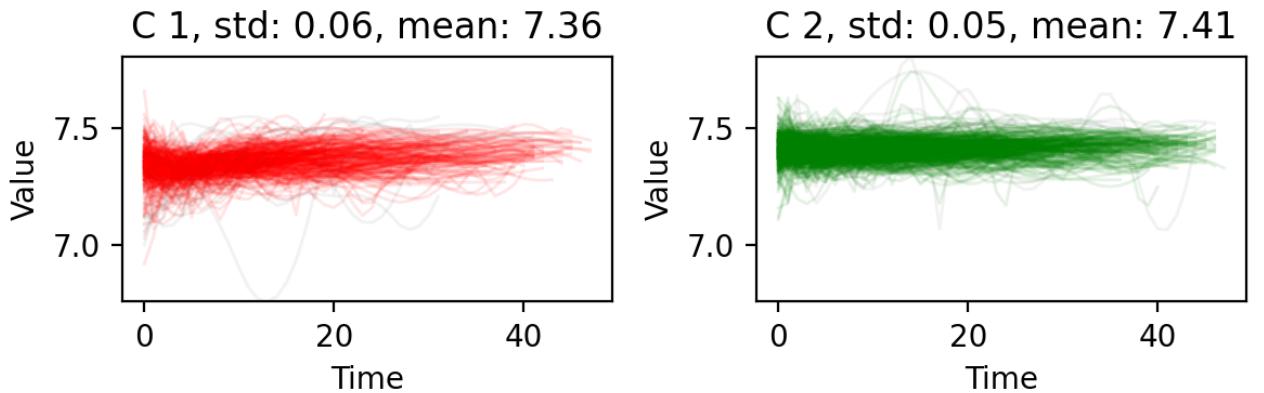
Creating stacked DF...

Cleaning

Creating stacked DF...

Cleaning

Arterial pH, Sil score: 0.23



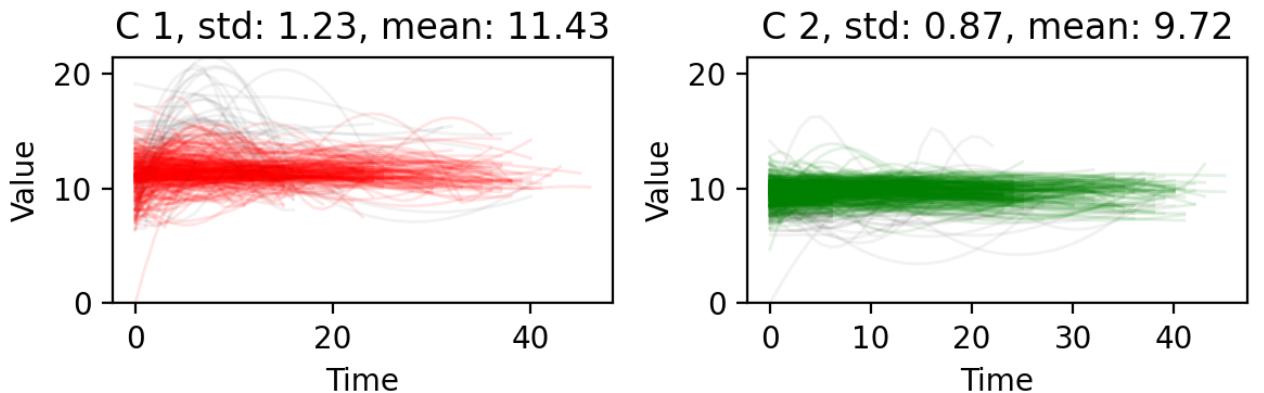
Creating stacked DF...

Cleaning

Creating stacked DF...

Cleaning

Hemoglobin, Sil score: 0.39



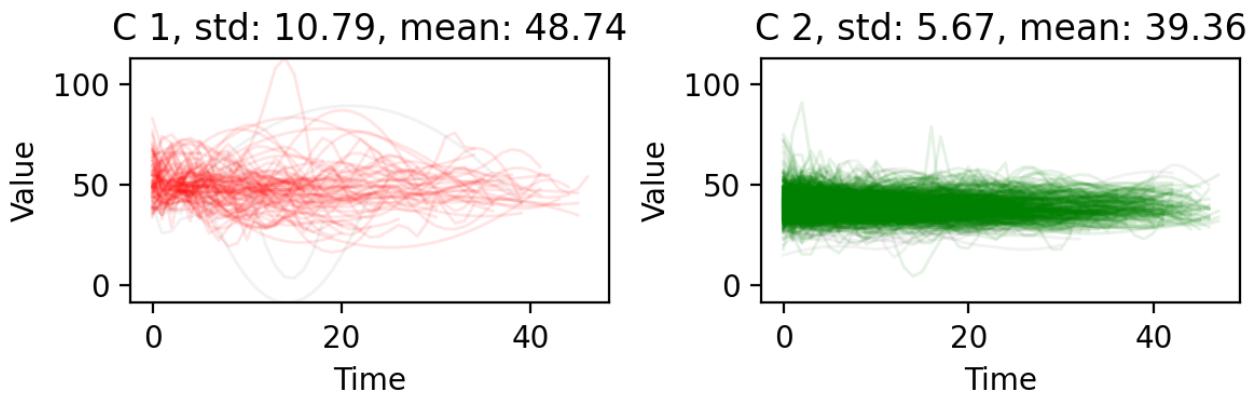
Creating stacked DF...

Cleaning

Creating stacked DF...

Cleaning

Arterial PaCO₂, Sil score: 0.46



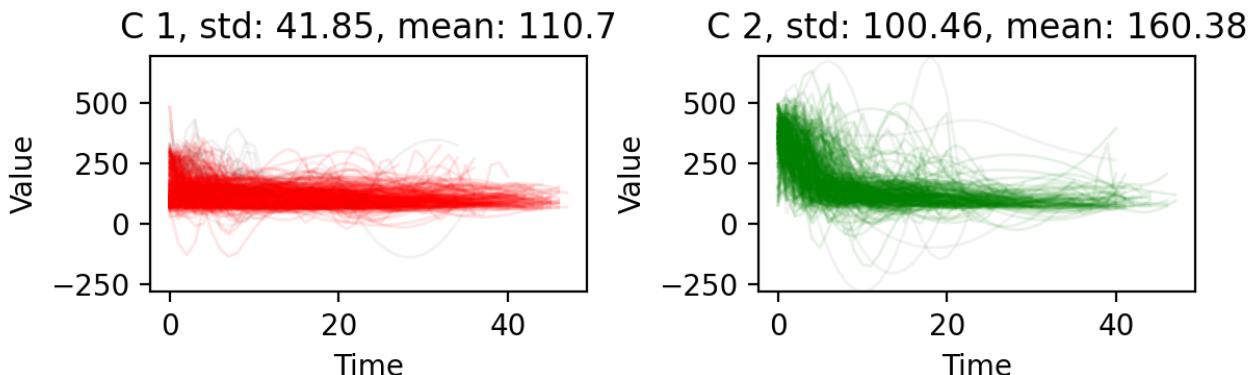
Creating stacked DF...

Cleaning

Creating stacked DF...

Cleaning

Arterial PaO₂, Sil score: 0.39



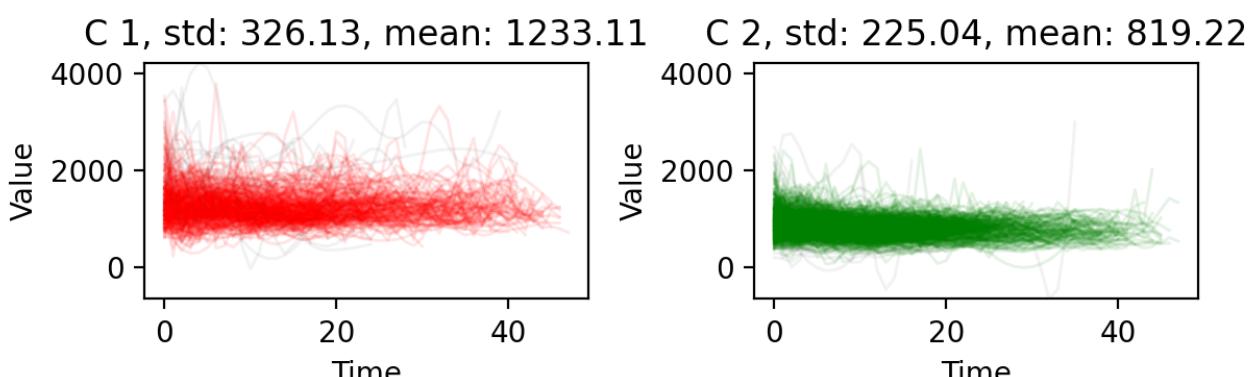
Creating stacked DF...

Cleaning

Creating stacked DF...

Cleaning

SVR, Sil score: 0.37



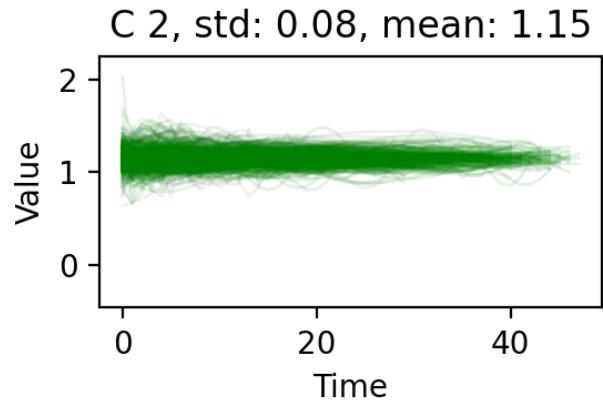
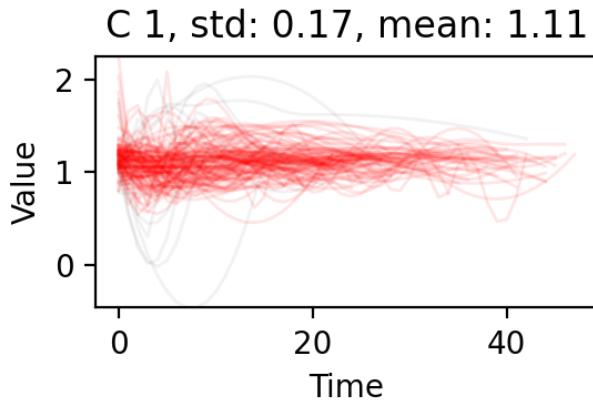
Creating stacked DF...

Cleaning

Creating stacked DF...

Cleaning

Ionized Calcium, Sil score: 0.39



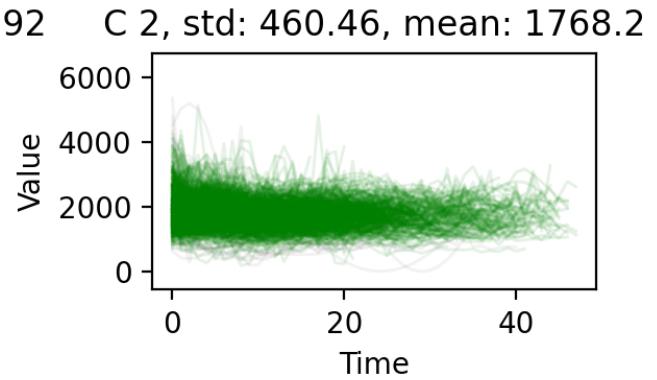
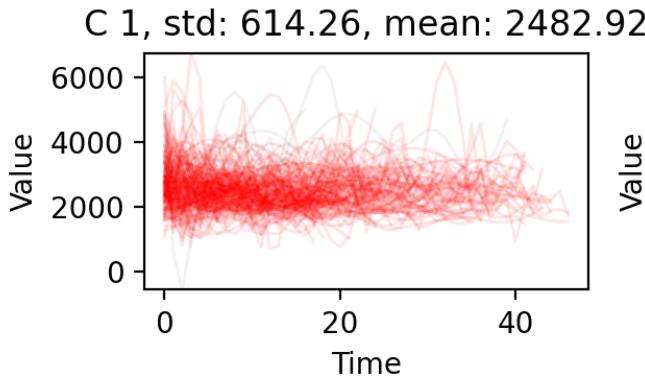
Creating stacked DF...

Cleaning

Creating stacked DF...

Cleaning

SVRI, Sil score: 0.39



	Feature	StdDev	Mean
0	Platelets	50.800181	218.674414
1	Platelets	33.859267	117.201712
2	Arterial BP [Diastolic]	10.429068	64.059164
3	Arterial BP [Diastolic]	8.256556	52.779701
4	Arterial BP [Systolic]	15.188490	109.887503
5	Arterial BP [Systolic]	19.990647	131.377239
6	Arterial BP Mean	11.497667	75.020675
7	Arterial BP Mean	21.071728	85.927426
8	CVP	4.422265	9.724471
9	CVP	4.852999	13.029135
10	Arterial pH	0.058331	7.358850
11	Arterial pH	0.046653	7.410690
12	Hemoglobin	1.228638	11.426426
13	Hemoglobin	0.868202	9.722798
14	Arterial PaCO2	10.791743	48.736232
15	Arterial PaCO2	5.671661	39.358229
16	Arterial PaO2	41.846969	110.704124
17	Arterial PaO2	100.458986	160.380584
18	SVR	326.134697	1233.109675
19	SVR	225.037165	819.221521

	Feature	StdDev	Mean
20	Ionized Calcium	0.170464	1.113496
21	Ionized Calcium	0.080157	1.145253
22	SVRI	614.258344	2482.924146
23	SVRI	460.460522	1768.198153

Helper code to combine the clustering graphs into one big graph for the report

```
In [ ]: # figdir = "./figures/"

# 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 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'):

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

        mapping = getMapping(metric, subset)

        newCol = str(name + " " + metric)

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

display(orderedDF.head())
```

Platelets	Arterial BP [Diastolic]	Arterial BP [Systolic]	Arterial BP Mean	CVP	Arterial pH	Hemoglobin	Arterial PaCO2	Arterial PaO2	SVR	Ionized Calcium	SVRI
-----------	----------------------------	---------------------------	---------------------	-----	----------------	------------	-------------------	------------------	-----	--------------------	------

PatientID

temporalClustering

	Platelets	Arterial BP [Diastolic]	Arterial BP [Systolic]	Arterial BP Mean	CVP	Arterial pH	Hemoglobin	Arterial PaCO2	Arterial PaO2	SVR	Ionized Calcium	SVRI
PatientID	0	1	1	1	0	1	1	0	1	1	1	1
0	0	1	0	0	0	0	1	1	1	0	0	1
0	1	1	1	0	0	1	1	1	1	0	1	1
0	1	0	0	0	0	1	0	0	1	0	1	0
0	1	0	0	0	0	1	0	0	1	1	1	1

	Arterial BP Mean_StdDev	Arterial BP Mean_Mean	Arterial BP [Diastolic]_StdDev	Arterial BP [Diastolic]_Mean	Arterial BP [Systolic]_StdDev	Arterial BP [Systolic]_Mean	Arterial PaCO2_StdDev	Arterial PaCO2_Mean
PatientID	0	0	0	0	0	1	1	0
0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	1	1	0
0	0	0	1	1	0	0	0	0
0	0	0	1	1	0	0	1	1

5 rows × 24 columns

◀	▶
---	---

Mapping low to high

```
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={0: 'very_low', 1: 'low', 2: 'medium', 3: 'high', 4: 'ver
display(mapped)
```

	Arterial BP Mean_StdDev_high	Arterial BP Mean_StdDev_low	Arterial BP Mean_Mean_high	Arterial BP Mean_Mean_low	Arterial BP [Diastolic]_StdDev_high	Arterial BP [Diastolic]_StdDev_low	Arterial BP [Diastolic]_Mean
PatientID	0	0	1	0	1	0	1
0	0	1	0	1	0	0	1
0	0	1	0	1	0	0	1
0	0	1	0	1	0	0	1
0	0	1	0	1	0	1	0
0	0	1	0	1	0	1	0
...
0	0	1	0	1	0	0	1
0	0	1	0	1	0	1	0
0	0	1	0	1	0	0	1
0	0	1	0	1	0	1	0
0	0	1	0	1	0	0	1

959 rows × 48 columns

◀	▶
---	---

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

```
In [ ]: pd.Series(targetSeries).value_counts()
```

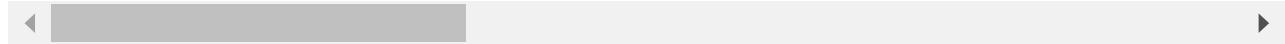
```
Out[ ]: 0    924
1     35
dtype: int64
```

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

```
display(mapped)
mapped.to_csv("./categorisedData/clusteredData.csv")
```

PatientID	Arterial BP Mean_StdDev_high	Arterial BP Mean_StdDev_low	Arterial BP Mean_Mean_high	Arterial BP Mean_Mean_low	Arterial BP [Diastolic]_StdDev_high	Arterial BP [Diastolic]_StdDev_low	Arterial BP [Diastolic]_
0	0	1	0	1	0	1	
0	0	1	0	1	0	1	
0	0	1	0	1	0	1	
0	0	1	0	1	1	0	
0	0	1	0	1	1	0	
...	
0	0	1	0	1	0	1	
0	0	1	0	1	1	0	
0	0	1	0	1	0	1	
0	0	1	0	1	1	0	
0	0	1	0	1	0	1	

959 rows × 49 columns



Exploring the differences in each class' features

```
In [ ]: minority = mapped[mapped['Mortality14Days'] == 1]
majority = mapped[mapped['Mortality14Days'] == 0]

display(mapped.describe())
display(minority.describe())
display(majority.describe())
```

	Arterial BP Mean_StdDev_high	Arterial BP Mean_StdDev_low	Arterial BP Mean_Mean_high	Arterial BP Mean_Mean_low	Arterial BP [Diastolic]_StdDev_high	Arterial BP [Diastolic]_StdDev_low	Arterial BP [Diastolic]_
count	959.000000	959.000000	959.000000	959.000000	959.000000	959.000000	959.000000
mean	0.080292	0.919708	0.080292	0.919708	0.363921	0.636079	
std	0.271886	0.271886	0.271886	0.271886	0.481377	0.481377	
min	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	
25%	0.000000	1.000000	0.000000	1.000000	0.000000	0.000000	
50%	0.000000	1.000000	0.000000	1.000000	0.000000	1.000000	
75%	0.000000	1.000000	0.000000	1.000000	1.000000	1.000000	
max	1.000000	1.000000	1.000000	1.000000	1.000000	1.000000	

8 rows × 49 columns



	Arterial BP Mean_StdDev_high	Arterial BP Mean_StdDev_low	Arterial BP Mean_Mean_high	Arterial BP Mean_Mean_low	Arterial BP [Diastolic]_StdDev_high	Arterial BP [Diastolic]_StdDev_low	Arterial BP [Diastolic]_
count	35.000000	35.000000	35.000000	35.000000	35.000000	35.000000	35.000000
mean	0.057143	0.942857	0.057143	0.942857	0.400000	0.600000	
std	0.235504	0.235504	0.235504	0.235504	0.49705	0.49705	
min	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	
25%	0.000000	1.000000	0.000000	1.000000	0.000000	0.000000	
50%	0.000000	1.000000	0.000000	1.000000	0.000000	1.000000	
75%	0.000000	1.000000	0.000000	1.000000	1.000000	1.000000	
max	1.000000	1.000000	1.000000	1.000000	1.000000	1.000000	

8 rows × 49 columns

	Arterial BP Mean_StdDev_high	Arterial BP Mean_StdDev_low	Arterial BP Mean_Mean_high	Arterial BP Mean_Mean_low	Arterial BP [Diastolic]_StdDev_high	Arterial BP [Diastolic]_StdDev_low	Arterial BP [Diastolic]_
count	924.000000	924.000000	924.000000	924.000000	924.000000	924.000000	924.000000
mean	0.081169	0.918831	0.081169	0.918831	0.362554	0.637446	
std	0.273242	0.273242	0.273242	0.273242	0.480998	0.480998	
min	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	
25%	0.000000	1.000000	0.000000	1.000000	0.000000	0.000000	
50%	0.000000	1.000000	0.000000	1.000000	0.000000	1.000000	
75%	0.000000	1.000000	0.000000	1.000000	1.000000	1.000000	
max	1.000000	1.000000	1.000000	1.000000	1.000000	1.000000	

8 rows × 49 columns